Starlight

Généré par Doxygen 1.8.9.1

Jeudi 23 Avril 2015 11 :07 :49

Table des matières

Chapitre 1

Page principale

Introduction

"Starlight est un petit jeu en deux dimensions se jouant sur une carte rectangulaire, comportant une source de lumière, émettant un rayon recti-ligne.Le but du jeuest d'atteindre une cible avec ledit rayon, en évitant les obstacles via notamment des miroirs réfléchissant la lumière."

Rules

Starlight est un puzzle à deux dimensions se jouant sur une carte rec-tangulaire. Le but du jeu est de dévier un rayon lumineux d'une source vers une cible en évitant certains obstacles. Plus particulièrement, on trouve les éléments suivants sur une carte.

- Une unique source : cet élément émet un rayon lumineux d'une lon- gueur d'onde donnée sous un certain angle.
- Une unique cible (ou destination) : cet élément doit être éclairé par un rayon lumineux pour remporter la partie.
- Un ensemble de miroirs : un miroir est un objet réfléchissant la lumière d'un seul côté suivant le schéma naturel de la réflexion de la lumière. Plus particulièrement, un rayon incident à un miroir sous un angle i sera réfléchi sous le même angle r , comme illustré 1 à la Figure 1.
- Un ensemble de murs : les murs ne réfléchissent pas la lumière. Tout rayon incident à un mur ne se propage pas, et " s'arrête " donc là où il y est incident.
- Un ensemble de lentilles. Les lentilles sont des objets transparents qui ne laissent passer un rayon lumineux que dans un certain intervalle de longueur d'onde [m ,n]. Si un rayon lumineux possède une longueur d'onde telle que m <= <= n, il traverse la lentille sans subir aucune modification. Sinon, la lentille stoppe le rayon (elle se comporte comme un mur).
- Un ensemble de cristaux : un cristal est un élément transparent qui modifie la longueur d'onde d'un rayon, en l'augmentant ou la dimi- nuant. Tout rayon qui traverse un cristal le traverse donc sans subir de modification de trajectoire, mais voit sa longueur d'onde modifiée.
- Un ensemble de bombes. Les bombes sont des objets qui, si éclairés, explosent et font automatiquement perdre la partie au joueur.
- Un ensemble de rayons. Initialement émis par la source du jeu, ils sont rectilignes et se réfléchissent sur les miroirs. Un rayon est donc un segment de droite. Sur la Figure 1, on voit donc deux rayons, le rayon P et le rayon Q. Un rayon possède également une autre caractéristique : sa longueur d'onde. La longueur d'onde d'un rayon permet de déterminer, comme mentionné ci-dessus, si oui ou non un rayon traverse une lentille. Elle est modifiée par un cristal.

Chapitre 2

Index des espaces de nommage

2.1 Liste des espaces de nommage

Liste de tous les espaces de nommage avec une brève description :

Chapitre 3

Index hiérarchique

3.1 Hiérarchie des classes

Cette liste d'héritage est classée approximativement par ordre alphabétique :

Chapitre 4

Index des classes

4.1 Liste des classes

Liste des classes, structures, unions et interfaces avec une brève description :

Chapitre 5

Index des fichiers

5.1 Liste des fichiers

Liste de tous les fichiers avec une brève description :

Chapitre 6

Documentation des espaces de nommage

6.1 Référence de l'espace de nommage levelFactory

Fonctions utilitaires permettant divers éléments du jeu à partir d'un fichier .lvl.

Fonctions

- Level * getLevelFromFile (std : :string)
  Permet d'obtenir une référence vers une nouvelle carte initialisée à partir d'un fichier .level.
- Source getSource (std : :ifstream &)
  Permet d' obtenir une source à partir d'un fichier .lvl déjà ouvert.
- Dest getDestination (std : :ifstream &)
  Permet d' obtenir une destination à partir d'un fichier .lvl déjà ouvert.
- Crystal getCrystal (std : :ifstream &)
  Permet d' obtenir un crystal à partir d'un fichier .lvl déjà ouvert.
- Lens getLens (std : :ifstream &)
  Permet d' obtenir une lentille à partir d'un fichier .lvl déjà ouvert.
- Wall getWall (std : :ifstream &)
  Permet d' obtenir un mur à partir d'un fichier .lvl déjà ouvert.
- Nuke getNuke (std : :ifstream &)
  Permet d' obtenir une bombe à partir d'un fichier .lvl déjà ouvert.
- Mirror getMirror (std : :ifstream &)
  Permet d' obtenir un mirroir à partir d'un fichier .lvl déjà ouvert.

6.1.1 Description détaillée

Fonctions utilitaires permettant divers éléments du jeu à partir d'un fichier .lvl.

6.1.2 Documentation des fonctions

6.1.2.1 Crystal levelFactory : :getCrystal (std : :ifstream &)

Permet d' obtenir un crystal à partir d'un fichier .lvl déjà ouvert.

Paramètres

mapFile	Fichier .lvl déjà ouvert.

Renvoie

Un cristal.

6.1.2.2 Dest levelFactory : :getDestination (std : :ifstream &)

Permet d' obtenir une destination à partir d'un fichier .lvl déjà ouvert.

Paramètres

mapFile	Fichier .lvl déjà ouvert.

Renvoie

Une destination.

6.1.2.3 Lens levelFactory : :getLens (std : :ifstream &)

Permet d' obtenir une lentille à partir d'un fichier .lvl déjà ouvert.

Paramètres

mapFile	Fichier .lvl déjà ouvert.

Renvoie

Une lentille.

6.1.2.4 Level∗ levelFactory : :getLevelFromFile (std : :string)

Permet d'obtenir une référence vers une nouvelle carte initialisée à partir d'un fichier .level.

Paramètres

mapFilePath	chemin vers le fichier .level.

Renvoie

une référence vers une nouvelle carte initialisée.

6.1.2.5 Mirror levelFactory : :getMirror (std : :ifstream &)

Permet d' obtenir un mirroir à partir d'un fichier .lvl déjà ouvert.

Paramètres

mapFile	Fichier .lvl déjà ouvert.

Renvoie

Un mirroir.

6.1.2.6 Nuke levelFactory : :getNuke (std : :ifstream &)

Permet d' obtenir une bombe à partir d'un fichier .lvl déjà ouvert.

Paramètres

mapFile	Fichier .lvl déjà ouvert.

Renvoie

Une bombe.

6.1.2.7 Source levelFactory : :getSource (std : :ifstream &)

Permet d' obtenir une source à partir d'un fichier .lvl déjà ouvert.

Paramètres

mapFile	Fichier .lvl déjà ouvert.

Renvoie

Une source.

6.1.2.8 Wall levelFactory : :getWall (std : :ifstream &)

Permet d' obtenir un mur à partir d'un fichier .lvl déjà ouvert.

Paramètres

mapFile	Fichier .lvl déjà ouvert.

Renvoie

Un mur.

6.2 Référence de l'espace de nommage utilities

Diverse fonctions utilitaires de géométrie.

Fonctions

- bool secondDegreeEquationSolver (double, double, double, double $*$, double $*$)
  Permet de trouver les racines (si elles existe) d'une fonction du deuxième degré de forme $ax + bx + c$.
- double radianAsDegree (const double)
  Permet de trouver l'angle en degré d'un angle en radian.
- double radianAsDegree0to360 (const double)
  Permet de trouver l'angle en degré, entre 0 et 360, d'un angle en radian.
- bool equals (const double, const double, const double=utilities : :EPSILON)
  Cette méthode permet de savoir si deux double sont égaux avec une marge d'erreur Epsilon passée en paramètre ou imposée par défaut à $\epsilon = 10^{-7}$.
- int round (const double)
  Cette méthode cast un double en int on l'ayant au préalable arrondi à l'unité la plus proche (0.5).
- bool greaterOrEquals (const double, const double, const double=utilities : :EPSILON)
  Cette méthode permet de vérifier l'inégalité $nb_1 \geq nb_2$ sur deux nombres réels avec une marge d'erreur Epsilon passée en paramètre ou imposée par défaut à $\epsilon = 10^{-7}$.
- bool lessOrEquals (const double, const double, const double=utilities : :EPSILON)
  Cette méthode permet de vérifier l'inégalité $nb_1 \leq nb_2$ sur deux nombres réels avec une marge d'erreur Epsilon passée en paramètre ou imposée par défaut à $\epsilon = 10^{-7}$.
- double degreeToRadian (const double)
  Cette méthode permet de transformer des degrés en radian.
- double slopeFromPoints (const Point &, const Point &)
  Permet de trouver la pente d'une droite formée par deux points.
- bool isHalfPiPlusNPi (const double)
  Permet de savoir si l'angle, en radian, vaut $\frac{\pi}{2} + n \cdot (2 \cdot \pi)$.
- double tan (const double)
  Permet d'avoir la valeur trigonométrique tangente d'un angle ou l'infini si $angle = \frac{\pi}{2} + n \cdot 2 \cdot \pi$.
- double absoluteAngle (const double)
  Permet d'avoir l'angle "absolu" de celui passé en paramètre, [0, PI_2].
- double inZeroTwoPi (const double)
  Permet de cadrer un angle dans un intervalle [0 ; 2PI[.

Variables

- const double PI {3.14159265358979323846}
  PI Représentation de la constante PI sur 26 décimales.
- const double PI_2 {1.57079632679489661923}

PI_2 Représentation de la constante PI/2 sur 26 décimales.
- const double PI_4 {0.78539816339744830961 6}
  PI_4 Représentation de la constante PI/4 sur 26 décimales.
- const double EPSILON {10E-7}
  EPSILON Représentation de la marge d'erreur maximale acceptée.
- const double INF {1./0.}
  INF Représente une division impossible.

6.2.1 Description détaillée

Diverse fonctions utilitaires de géométrie.

6.2.2 Documentation des fonctions

6.2.2.1 double utilities : :absoluteAngle (const double)

Permet d'avoir l'angle "absolu" de celui passé en paramètre, [0, PI_2].

Renvoie

L'angle absolu de celui passé en paramètre.

6.2.2.2 double utilities : :degreeToRadian (const double)

Cette méthode permet de transformer des degrés en radian.

Renvoie

La valeur en radian de l'angle en degré passé en paramètre.

6.2.2.3 bool utilities : :equals (const double , const double , const double = utilities::EPSILON)

Cette méthode permet de savoir si deux double sont égaux avec une marge d'erreur Epsilon passée en paramètre ou imposée par défaut à $\epsilon = 10^{-7}$.

Paramètres

nb1	Un réel.
nb2	Un réel.
epsilon	Niveau de précision souhaitée permettant de justifier l'égalité ou $\epsilon = 10^{-7}$ par défaut.

Renvoie

`true` Si les deux nombres sont égaux avec la précision souhaitée.

6.2.2.4 bool utilities : :greaterOrEquals (const double , const double , const double = utilities::EPSILON)

Cette méthode permet de vérifier l'inégalité $nb_1 \geq nb_2$ sur deux nombres réels avec une marge d'erreur Epsilon passée en paramètre ou imposée par défaut à $\epsilon = 10^{-7}$.

Paramètres

nb1	Un nombre réels.

nb2	Un nombre réels.

Renvoie

`true` Si l'inégalité $nb_1 \geq nb_2$ est vérifiée.

6.2.2.5 double utilities : :inZeroTwoPi (const double)

Permet de cadrer un angle dans un intervalle [0 ; 2PI[.

Renvoie

L'angle passé en paramètre dans l'intervalle [0 ; 2PI[

6.2.2.6 bool utilities : :isHalfPiPlusNPi (const double)

Permet de savoir si l'angle, en radian, vaut $\frac{\pi}{2} + n \cdot (2 \cdot \pi)$.

Renvoie

`true` Si $angle = \frac{\pi}{2} + n \cdot (2 \cdot \pi)$

6.2.2.7 bool utilities : :lessOrEquals (const double , const double , const double = utilities::EPSILON)

Cette méthode permet de vérifier l'inégalité $nb_1 \leq nb_2$ sur deux nombres réels avec une marge d'erreur Epsilon passée en paramètre ou imposée par défaut à $\epsilon = 10^{-7}$.

Paramètres

nb1	Un nombre réels.
nb2	Un nombre réels.

Renvoie

`true` Si l'inégalité $nb_1 \geq nb_2$ est vérifiée.

6.2.2.8 double utilities : :radianAsDegree (const double)

Permet de trouver l'angle en degré d'un angle en radian.

Paramètres

alpha	Un angle en radian.

Renvoie

L'angle exprimé en degré.

6.2.2.9 double utilities : :radianAsDegree0to360 (const double)

Permet de trouver l'angle en degré, entre 0 et 360, d'un angle en radian.

Le calcul permet d'encadrer les cas où
- $\alpha < 0$
- $\alpha \geq 2\pi$
- $0 \leq \alpha < 2\pi$

Paramètres

alpha	Un angle en radian.

Renvoie

L'angle exprimé en degré dans l'intervalle $[0, 360[$

6.2.2.10 int utilities : :round (const double)

Cette méthode cast un double en int on l'ayant au préalable arrondi à l'unité la plus proche (0.5).

Renvoie

Le nombre arrondi.

Référencé par Level : :getHeight(), et Level : :getWidth().

6.2.2.11 bool utilities : :secondDegreeEquationSolver (double , double , double , double * , double *)

Permet de trouver les racines (si elles existe) d'une fonction du deuxième degré de forme $ax + bx + c$.

Paramètres

a	Paramètre de x².
b	Paramètre de x.
c	Terme indépendant.
rad1	Pointeur vers le conteneur de la valeur de la racine obtenue avec delta positif (non utilisé s'il n'existe pas de racines).
rad2	Pointeur vers le conteneur de la valeur de la racine obtenue avec delta négatif (non utilisé s'il n'existe pas de racines).

Renvoie

`true` S'il existe des racines.

6.2.2.12 double utilities : :slopeFromPoints (const Point & , const Point &)

Permet de trouver la pente d'une droite formée par deux points.

Paramètres

p1	Un point.
p2	Un point.

Renvoie

La pente de l'équation de droite passant par ces deux points.

6.2.2.13 double utilities : :tan (const double)

Permet d'avoir la valeur trigonométrique tangente d'un angle ou l'infini si $angle = \frac{\pi}{2} + n \cdot 2 \cdot \pi$.

Renvoie

La tangente de l'angle ou l'infini.

6.2.3 Documentation des variables

6.2.3.1 const double utilities : :EPSILON {10E-7}

EPSILON Représentation de la marge d'erreur maximale acceptée.

Définition à la ligne 30 du fichier utilities.hpp.

6.2.3.2 const double utilities : :INF {1./0.}

INF Représente une division impossible.

Définition à la ligne 35 du fichier utilities.hpp.

6.2.3.3 const double utilities : :PI {3.14159265358979323846}

PI Représentation de la constante PI sur 26 décimales.

Définition à la ligne 15 du fichier utilities.hpp.

6.2.3.4 const double utilities : :PI_2 {1.57079632679489661923}

PI_2 Représentation de la constante PI/2 sur 26 décimales.

Définition à la ligne 20 du fichier utilities.hpp.

6.2.3.5 const double utilities : :PI_4 {0.78539816339744830962}

PI_4 Représentation de la constante PI/4 sur 26 décimales.

Définition à la ligne 25 du fichier utilities.hpp.

6.3 Référence de l'espace de nommage viewUtilities

Divers fonctions utilitaires nécessaires aux vues.

Fonctions

- QPointF toQPoint (const Point &)
  Permet de transformer un point en QPointF.
- QRectF toQRectF (const Rectangle &)
  Permet de transformer un rectangle en QRectF.
- QRectF toQRectF (const Ellipse &)
  Permet de représenter une ellipse, à partir du rectangle qui lui est circonscrit, en un QRectF.
- QGraphicsLineItem ∗ getLine (const Point &, const Point &, const QColor &, const int)
  Permet de générer une QGraphicsLine à partir des deux points délimitant un segment de droite.
- QGraphicsRectItem ∗ getRect (const Rectangle &, const QColor &, const int)
  Permet de générer un QGraphicsRectItem représentant le rectangle passé en paramètre.
- QGraphicsEllipseItem ∗ getEllipse (const Ellipse &, const QColor &, const int)
  Permet de générer un QGraphicsEllipseItem représentant l'ellipse passée en paramètre.
- QColor waveLengthToColor (const Ray &, const double=0.8)
  Permet de créé une QColor au format RGB selon la longueur d'onde passée en paramètre.

6.3.1 Description détaillée

Divers fonctions utilitaires nécessaires aux vues.

6.3.2 Documentation des fonctions

6.3.2.1 QGraphicsEllipseItem* viewUtilities : :getEllipse (const Ellipse & , const QColor & , const int)

Permet de générer un QGraphicsEllipseItem représentant l'ellipse passée en paramètre.

Paramètres

ellipse	Une ellipse devant être représentée.
color	La couleur de cette ellipse.
width	L'épaisseur du trait représentant cette ellipse.

Renvoie

Le QGraphicsEllipseItem représentant cette ellipse.

6.3.2.2 QGraphicsLineItem* viewUtilities : :getLine (const Point & , const Point & , const QColor & , const int)

Permet de générer une QGraphicsLine à partir des deux points délimitant un segment de droite.

Paramètres

start	Le point de départ du segment de droite.
end	Le point d'arrivé du segment de droite.
color	La couleur que ce segment doit prendre.
width	L'épaisseur du trait de ce segment de droite.

Renvoie

Le QGraphicsLineItem représentant ce segment de droite.

6.3.2.3 QGraphicsRectItem* viewUtilities : :getRect (const Rectangle & , const QColor & , const int)

Permet de générer un QGraphicsRectItem représentant le rectangle passé en paramètre.

Paramètres

rectangle	Le rectangle devant être représenté.
color	La couleur de ce rectangle.
width	L'épaisseur du trait représentant ce rectangle.

Renvoie

Le QGraphicsRectItem représentant ce rectangle.

6.3.2.4 QPointF viewUtilities : :toQPoint (const Point &)

Permet de transformer un point en QPointF.

Renvoie

La représentation QPointF du Point passé en paramètre.

Voir également

Point

6.3.2.5 QRectF viewUtilities : :toQRectF (const Rectangle &)

Permet de transformer un rectangle en QRectF.

Renvoie

La représentation QRectF du rectangle passé en paramètre.

Voir également

Rectangle

6.3.2.6 QRectF viewUtilities : :toQRectF (const Ellipse &)

Permet de représenter une ellipse, à partir du rectangle qui lui est circonscrit, en un QRectF.

Renvoie

Le QRectF circonscrit à l'ellipse.

Voir également

Ellipse

6.3.2.7 QColor viewUtilities : :waveLengthToColor (const Ray & , const double = 0.8)

Permet de créé une QColor au format RGB selon la longueur d'onde passée en paramètre.

Cette méthode se base sur le spectre lumineux visible en représentant les U.V. comme du noir.

Renvoie

La QColor représentant la longueur d'onde d'un rayon.

Voir également

http://www.physics.sfasu.edu/astro/color/spectra.html

Chapitre 7

Documentation des classes

7.1 Référence de la classe Crystal

Cette classe amplifie les cristaux utilisés dans le jeu.

`#include <crystal.hpp>`

Fonctions membres publiques

- Crystal (const Point &, const double, const int)
  Instancier un cristal.
- int getAmplifier () const
  Retourne le modifieur de longueur d'onde du cristal.
- double getRadius () const
  Retourne la longueur du rayon du cristal.
- void reactToRay (Ray)
  Cette méthode est lancé lorsque le miroir courant est exposé à un rayon.
- Point * includeRay (const Ray &) const
  Renseigne si le crystal est dans la trajectoire du rayon.
- bool operator== (const Crystal &) const
  Permet de savoir si deux cristaux sont les même.
- bool operator != (const Crystal &) const
  Permet de savoir si deux cristaux sont différents.

Attributs privés

- int amplifier
  Le modificateur de longueur d'onde agissant sur un rayon passant dans ce cristal.

Membres hérités additionnels

7.1.1 Description détaillée

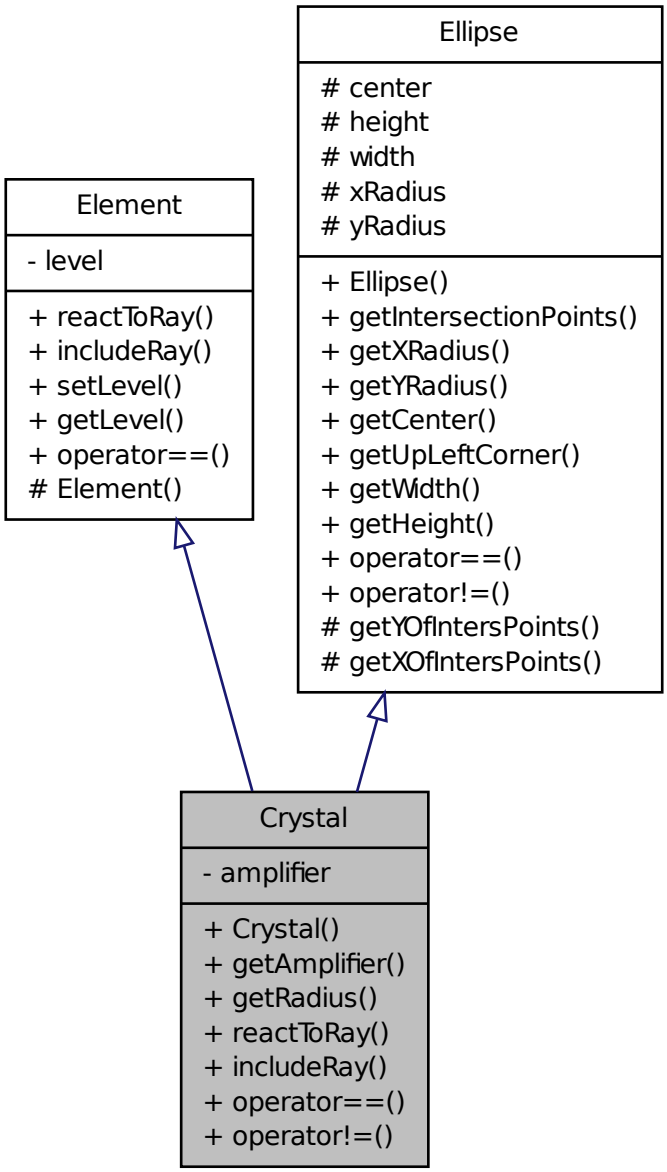
Cette classe amplifie les cristaux utilisés dans le jeu.

Un cristal est un objet circulaire centré en un point, et d'un certain rayon.

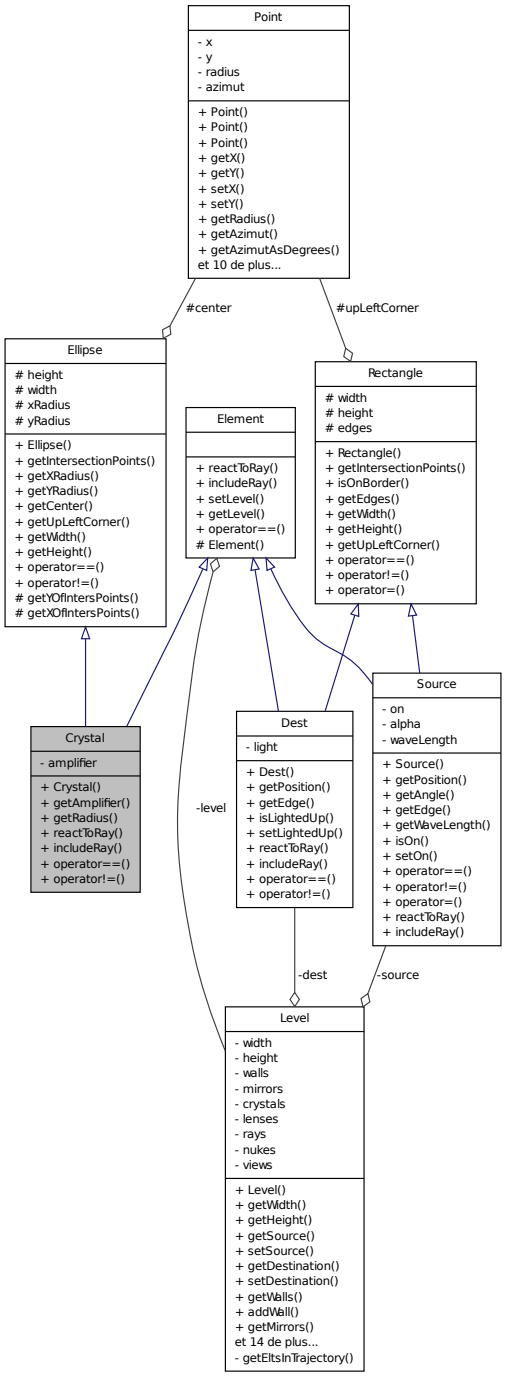
Un rayon lumineux passant à travers un crystal verra sa longueur d'onde modifiée (en l'augmentant ou en la diminuant d'une certaine valeur) mais pas sa trajectoire.

Définition à la ligne 23 du fichier crystal.hpp.

Graphe d'héritage de Crystal :



Graphe de collaboration de Crystal :



7.1.2 Documentation des constructeurs et destructeur

7.1.2.1 Crystal : :Crystal (const Point & , const double , const int)

Instancier un cristal.

– centré au point donné
– avec un rayon donné

– et un amplifier de longueur d'ondes donné

,

ce crystal modifie la longueur d'onde du rayon le traversant en suivant la règle suivante : si la longueur d'onde modifié sort de l'intervalle [longueur d'onde minimale, longueur d'onde maximale] alors elle ne sera pas appliquée.

Paramètres

point	Le point du centre du cristal.
radius	Le rayon du cristal.
amplifieur	le modificateur de longueur d'onde du cristal.

7.1.3 Documentation des fonctions membres

7.1.3.1 int Crystal : :getAmplifier ( ) const [inline]

Retourne le modifieur de longueur d'onde du cristal.

Renvoie

le modifieur de longueur d'onde du cristal

Définition à la ligne 110 du fichier crystal.hpp.

Références amplifier.

```
111 {
112     return this->amplifier;
113 }
```

7.1.3.2 double Crystal : :getRadius ( ) const

Retourne la longueur du rayon du cristal.

Renvoie

la longueur du rayon du cristal

7.1.3.3 Point* Crystal : :includeRay (const Ray &) const [virtual]

Renseigne si le crystal est dans la trajectoire du rayon.

Paramètres

ray	Un rayon.

Renvoie

Un pointeur vers le point d'intersection (le plus éloigné) avec le rayon s'il existe un pointeur null sinon.

Implémente Element.

7.1.3.4 bool Crystal : :operator != (const Crystal &) const

Permet de savoir si deux cristaux sont différents.

Renvoie

`true` si deux cristaux sont différents.

7.1.3.5 bool Crystal : :operator== (const Crystal &) const

Permet de savoir si deux cristaux sont les même.

Renvoie

`true` si deux cristaux sont les même.

7.1.3.6 void Crystal : :reactToRay (Ray) [virtual]

Cette méthode est lancé lorsque le miroir courant est exposé à un rayon.

Il va communiquer au niveau le nouveau rayon sortant du cristal.

Paramètres

ray	Un rayon percutant le miroir.

Implémente Element.

7.1.4 Documentation des données membres

7.1.4.1 int Crystal : :amplifier [private]

Le modificateur de longueur d'onde agissant sur un rayon passant dans ce cristal.

Définition à la ligne 31 du fichier crystal.hpp.

Référencé par getAmplifier().

La documentation de cette classe a été générée à partir du fichier suivant :
– model/elements/crystal.hpp

7.2 Référence de la classe Dest

Cette classe modélise la destination utilisée dans le jeu.

`#include <dest.hpp>`

Fonctions membres publiques

– Dest (const Point &, const int)
  Intancie une destination, de position et rayon donné.
– const Point & getPosition () const
  Retourne la position du coin supérieur gauche du carré modélisant la destination.
– int getEdge () const
  Retourne la longueur du côté du carré.
– bool isLightedUp () const
  Permet de savoir si la destination est éclairée et donc si le jeu est terminé.
– void setLightedUp (const bool)
  Permet de changer l'état d'illumination de la destination.
– void reactToRay (Ray)
  Cette méthode est lancé lorsque la destination courant est exposé à un rayon.
– Point ∗ includeRay (const Ray &) const
  Renseigne si la destination est dans la trajectoire du rayon.
– bool operator== (const Dest &) const
  Permet de savoir si deux destinations sont les même.
– bool operator != (const Dest &) const
  Permet de savoir si deux destinations sont différentes.

Attributs privés

– bool light

Membres hérités additionnels

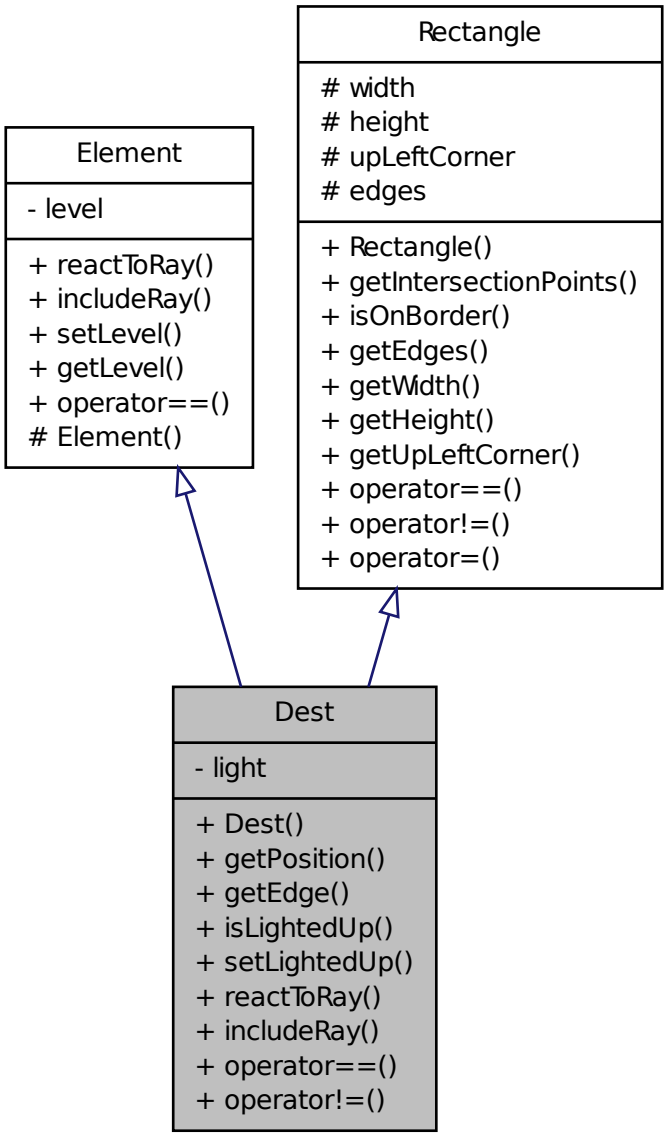
7.2.1 Description détaillée

Cette classe modélise la destination utilisée dans le jeu.

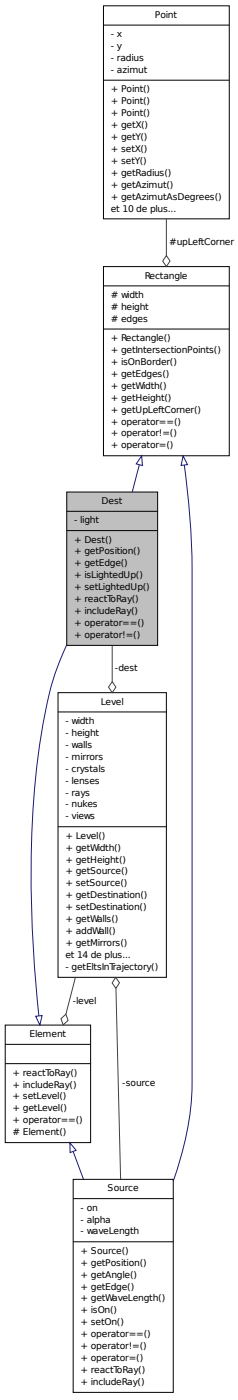
Une destination est un objet carré qui, quand traversé par un rayon lumineux, fait remporter la partie au joueur.

Définition à la ligne 17 du fichier dest.hpp.

Graphe d'héritage de Dest :



Graphe de collaboration de Dest :



7.2.2 Documentation des constructeurs et destructeur

7.2.2.1 Dest : :Dest (const Point & , const int)

Intancie une destination, de position et rayon donné.

Paramètres

position	Le coin supérieur gauche du carré modélisant la destination.
edge	La longueur du côté du carré.

7.2.3 Documentation des fonctions membres

7.2.3.1 int Dest : :getEdge ( ) const [inline]

Retourne la longueur du côté du carré.

Renvoie

La longueur du côté du carré.

Définition à la ligne 110 du fichier dest.hpp.

Références Rectangle : :height.

```
111 {
112     return this->height;
113 }
```

7.2.3.2 const Point & Dest : :getPosition ( ) const [inline]

Retourne la position du coin supérieur gauche du carré modélisant la destination.

Renvoie

La position de la destination.

Définition à la ligne 105 du fichier dest.hpp.

Références Rectangle : :upLeftCorner.

```
106 {
107     return this->upLeftCorner;
108 }
```

7.2.3.3 Point* Dest : :includeRay (const Ray &) const [virtual]

Renseigne si la destination est dans la trajectoire du rayon.

Paramètres

ray	Le rayon.

Renvoie

`true` Si la destination se trouve dans la trajectoire du rayon entré en paramètre.

Implémente Element.

7.2.3.4 bool Dest : :isLightedUp ( ) const [inline]

Permet de savoir si la destination est éclairée et donc si le jeu est terminé.

Renvoie

`true` Si la destination est illuminée.

Définition à la ligne 115 du fichier dest.hpp.

Références light.

```
116 {
117     return this->light;
118 }
```

7.2.3.5 bool Dest : :operator != (const Dest &) const

Permet de savoir si deux destinations sont différentes.

Renvoie

`true` Si les destinations sont différentes.

7.2.3.6 bool Dest : :operator== (const Dest &) const

Permet de savoir si deux destinations sont les même.

Renvoie

`true` Si les destinations sont les même.

7.2.3.7 void Dest : :reactToRay (Ray) [virtual]

Cette méthode est lancé lorsque la destination courant est exposé à un rayon.

Elle va s'exposer comme illuminée.

Paramètres

ray	Un rayon percutant la destination.

Implémente Element.

7.2.3.8 void Dest : :setLightedUp (const bool)

Permet de changer l'état d'illumination de la destination.

Paramètres

Le	nouvelle état d'illumination de la destination.

7.2.4 Documentation des données membres

7.2.4.1 bool Dest : :light [private]

Définition à la ligne 22 du fichier dest.hpp.

Référencé par isLightedUp().

La documentation de cette classe a été générée à partir du fichier suivant :
– model/elements/dest.hpp

7.3 Référence de la classe Element

Un élément est un composant du jeu se devant de communiquer son état au niveau le gérant.

`#include <element.hpp>`

Fonctions membres publiques

- virtual void reactToRay (Ray)=0
  Réaction à l'exposition d'un rayon.
- virtual Point * includeRay (const Ray &) const =0
  Renseigne si l'élément est dans la trajectoire du rayon.
- void setLevel (Level *)
  Permet de modifier le level auquel appartient l'élément.
- Level * getLevel ()
  Permet d'obtenir un pointeur sur le niveau auquel appartient l'élément.
- bool operator== (const Element &) const
  Compare deux éléments pour savoir si ils pointent vers le même niveau.

Fonctions membres protégées

- Element ()=default
  Constructeur par défaut, en visibilité protected permettant d'éviter une tentative d'instanciation de cette classe abstraite.

Attributs privés

- Level * level {nullptr}
  Le niveau lié à un élément.

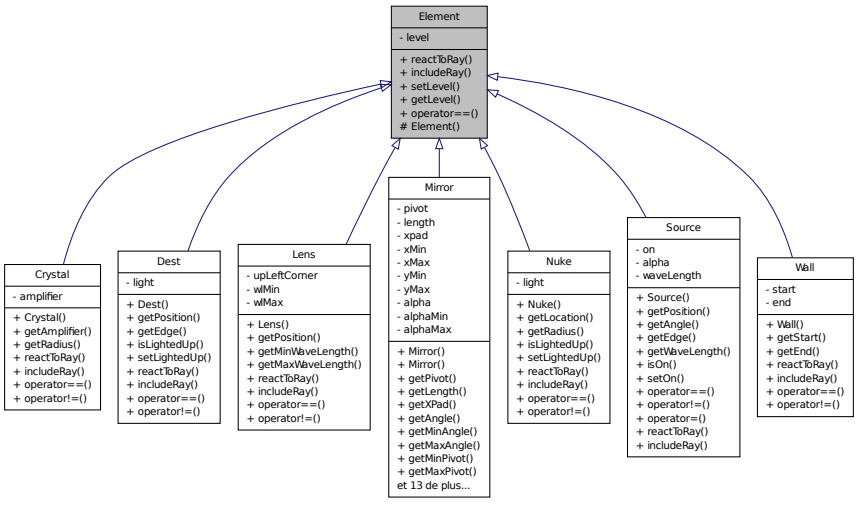
7.3.1 Description détaillée

Un élément est un composant du jeu se devant de communiquer son état au niveau le gérant.

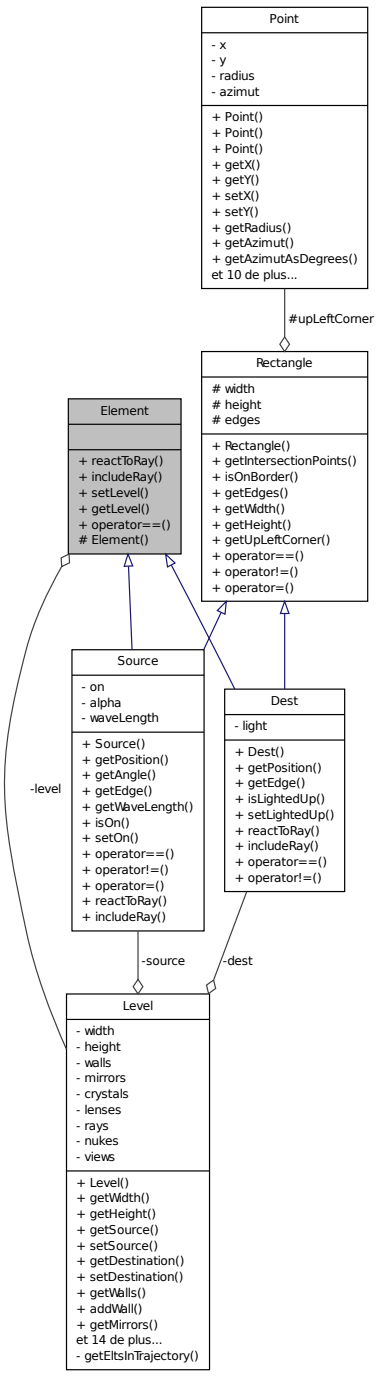
Cette pratique permet au niveau d'écouter les actions à éffectuer dicter par l'élément.

Définition à la ligne 13 du fichier element.hpp.

Graphe d'héritage de Element :



Graphe de collaboration de Element :



7.3.2 Documentation des constructeurs et destructeur

7.3.2.1 Element : :Element ( ) [protected], [default]

Constructeur par défaut, en visibilité protected permettant d'éviter une tentative d'instanciation de cette classe abstraite.

7.3.3 Documentation des fonctions membres

7.3.3.1 Level * Element : :getLevel ( ) [inline]

Permet d'obtenir un pointeur sur le niveau auquel appartient l'élément.

Renvoie

un pointeur vers le niveau auquel appartient l'élément.

Définition à la ligne 71 du fichier element.hpp.

Références level.

```
72 {
73     return this->level;
74 }
```

7.3.3.2 virtual Point* Element : :includeRay (const Ray &) const [pure virtual]

Renseigne si l'élément est dans la trajectoire du rayon.

Paramètres

ray	Le rayon.

Renvoie

`true` Si l'élément se trouve dans la trajectoire du rayon entré en paramètre.

Implémenté dans Mirror, Source, Lens, Crystal, Dest, Nuke, et Wall.

7.3.3.3 bool Element : :operator== (const Element &) const

Compare deux éléments pour savoir si ils pointent vers le même niveau.

Renvoie

`true` Si les deux éléments sont liés au même niveau.

7.3.3.4 virtual void Element : :reactToRay (Ray) [pure virtual]

Réaction à l'exposition d'un rayon.

Paramètres

ray	Le rayon.

Implémenté dans Mirror, Source, Lens, Crystal, Dest, Nuke, et Wall.

7.3.3.5 void Element : :setLevel (Level *)

Permet de modifier le level auquel appartient l'élément.

Paramètres

nouveau	level auquel appartient l'élément.

7.3.4 Documentation des données membres

7.3.4.1 Level* Element : :level {nullptr} [private]

Le niveau lié à un élément.

Définition à la ligne 20 du fichier element.hpp.

Référencé par getLevel().

La documentation de cette classe a été générée à partir du fichier suivant :
– model/elements/element.hpp

7.4 Référence de la classe Ellipse

Représente un cercle sous la forme ; $circle \equiv x^2/xRadius + y^2/yRadius = 1$.

`#include <ellipse.hpp>`

Fonctions membres publiques

– Ellipse (double, double, const Point &)
Permet de construire une nouvelle ellipse initialisée.
– std : :vector< Point > getIntersectionPoints (const Line &) const
Permet d'obtenir les points d'intersection entre le cercle et la droite entrée en paramètre.
– double getXRadius () const
Permet d'obtenir la valeur du ratio de largeur de l'ellipse.
– double getYRadius () const
Permet d'obtenir la valeur du ratio de hauteur de l'ellipse.
– Point getCenter () const
Permet d'obtenir le centre de l'ellipse.
– Point getUpLeftCorner () const
Permet d'obtenir le coin supérieur gauche du rectangle entourant l'ellipse.
– double getWidth () const
Permet d'obtenir la largeur de l'ellipse.
– double getHeight () const
Permet d'obtenir la hauteur de l'ellipse.
– bool operator== (const Ellipse &) const
Permet de savoir si deux Ellipse sont les mêmes.
– bool operator != (const Ellipse &) const
Permet de savoir si deux Ellipse sont différentes.

Fonctions membres protégées

– bool getYOfIntersPoints (const double, double *, double *) const
Permet d'obtenir les valeurs des ordonnées des points d'intersection entre l'ellipse et une droite verticale dont l'abscisse est entrée en paramètre.
– bool getXOfIntersPoints (const double, const double, double *, double *) const
Permet d'obtenir les abscisses des points d'intersection entre une droite non-verticale dont la pente et le terme indépendant de l'équation sont entrés en paramètre.

Attributs protégés

– Point center
La position du centre de l'ellipse.
– double height
La hauteur du rectangle circonscrit à l'ellipse.
– double width

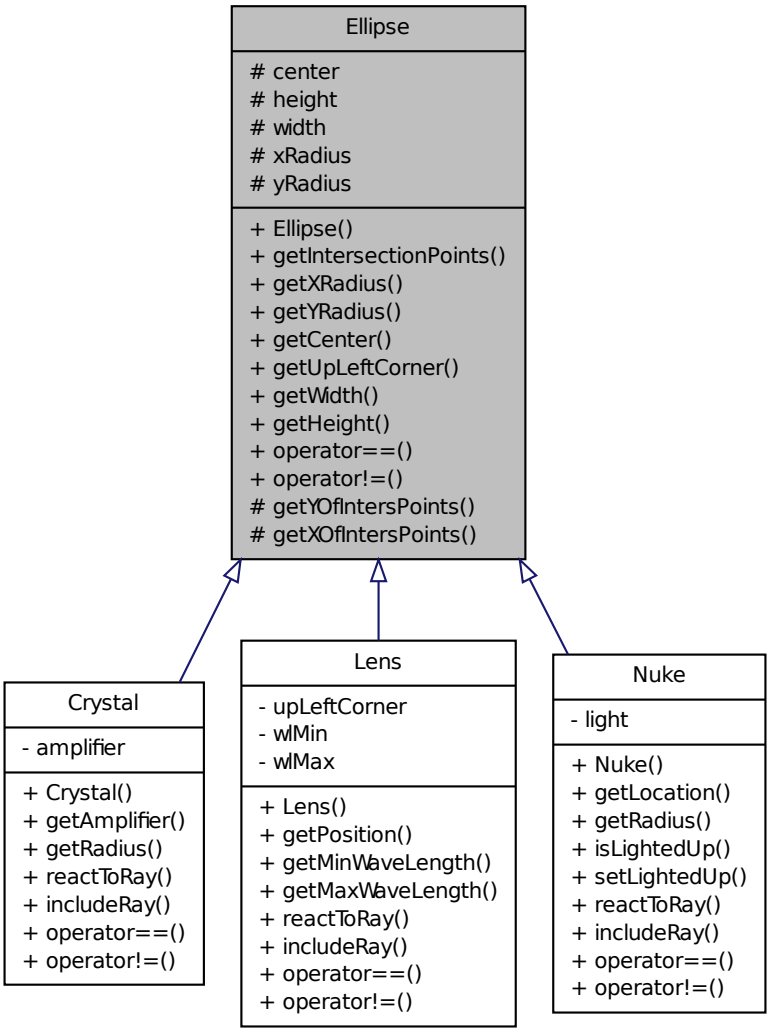
La largeur du rectangle circonscrit à l'ellipse.
– double xRadius
La valeur de la demi hauteur au carré.
– double yRadius
La valeur de la demi largeur au carré.

7.4.1 Description détaillée

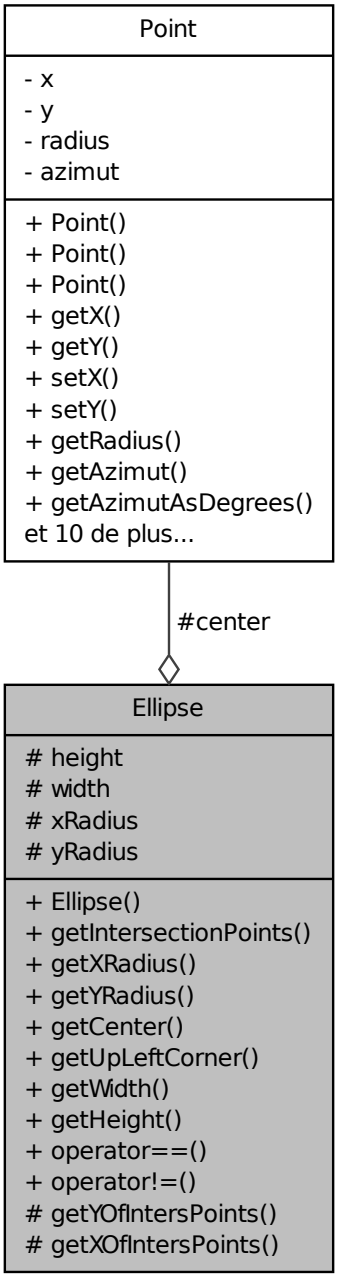
Représente un cercle sous la forme ; $circle \equiv x^2/xRadius + y^2/yRadius = 1$.

Définition à la ligne 15 du fichier ellipse.hpp.

Graphe d'héritage de Ellipse :



Graphe de collaboration de Ellipse :



7.4.2 Documentation des constructeurs et destructeur

7.4.2.1 Ellipse : :Ellipse (double , double , const Point &)

Permet de construire une nouvelle ellipse initialisée.

Paramètres

xRadius	Valeur du ratio de largeur de l'ellipse.
yRadius	Valeur du ratio de hauteur de l'ellipse.
center	Point du centre de l'ellipse.

7.4.3 Documentation des fonctions membres

7.4.3.1 Point Ellipse : :getCenter ( ) const [inline]

Permet d'obtenir le centre de l'ellipse.

Renvoie

Le centre de l'ellipse.

Définition à la ligne 192 du fichier ellipse.hpp.

Références center.

```
193 {
194     return this->center;
195 }
```

7.4.3.2 double Ellipse : :getHeight ( ) const [inline]

Permet d'obtenir la hauteur de l'ellipse.

Renvoie

La hauteur de l'ellipse.

Définition à la ligne 177 du fichier ellipse.hpp.

Références height.

Référencé par Nuke : :getRadius().

```
178 {
179     return this->height;
180 }
```

7.4.3.3 std : :vector<Point> Ellipse : :getIntersectionPoints (const Line &) const

Permet d'obtenir les points d'intersection entre le cercle et la droite entrée en paramètre.

Paramètres

line	droite dont on désire obtenir les points d'intersection avec l'éllipse.

Renvoie

Un vecteur contenant les points d'intersections entre le cercle et la droite entrée en paramètre.

7.4.3.4 Point Ellipse : :getUpLeftCorner ( ) const

Permet d'obtenir le coin supérieur gauche du rectangle entourant l'ellipse.

Renvoie

le coin supérieur gauche du rectangle entourant l'ellipse.

7.4.3.5 double Ellipse : :getWidth ( ) const [inline]

Permet d'obtenir la largeur de l'ellipse.

Renvoie

La largeur de l'ellipse.

Définition à la ligne 172 du fichier ellipse.hpp.

Références width.

```
173 {
174     return this->width;
175 }
```

7.4.3.6 bool Ellipse : :getXOfIntersPoints (const double , const double , double * , double *) const [protected]

Permet d'obtenir les abscisses des points d'intersection entre une droite non-verticale dont la pente et le terme indépendant de l'équation sont entrés en paramètre.

Paramètres

slope	Pente de la droite dont on désire les abscisses des points d'intersection avec l'ellipse.
lineIT	Terme indépendant de l'équation de la droite dont on désire les abscisses des points d'intersection avec l'ellipse.
x1	Conteneur de la valeur de l'abscisse du premier point d'intersection (non utilisé s'il n'existe pas de point d'intersection).
x2	Conteneur de la valeur de l'abscisse du deuxième point d'intersection (non utilisé s'il n'existe pas de point d'intersection).

Renvoie

`true` Si il existe des points d'intersection entre la droite et l'ellipse.

7.4.3.7 double Ellipse : :getXRadius ( ) const [inline]

Permet d'obtenir la valeur du ratio de largeur de l'ellipse.

Renvoie

La valeur du ratio de largeur de l'ellipse.

Définition à la ligne 182 du fichier ellipse.hpp.

Références xRadius.

```
183 {
184     return this->xRadius;
185 }
```

7.4.3.8 bool Ellipse : :getYOfIntersPoints (const double , double * , double *) const [protected]

Permet d'obtenir les valeurs des ordonnées des points d'intersection entre l'ellipse et une droite verticale dont l'abscisse est entrée en paramètre.

Paramètres

xValue	Abscisse de la droite verticale dont on désire les ordonnées des points d'intersection avec l'ellipse.
y1	Conteneur de la valeur de l'ordonnée du premier point d'intersection (non utilisé s'il n'existe pas de point d'intersection).
y2	Conteneur de la valeur de l'ordonne du deuxième point d'intersection (non utilisé s'il n'existe pas de point d'intersection).

Renvoie

`true` Si il existe des points d'intersection entre la droite et l'éllipse.

7.4.3.9 double Ellipse : :getYRadius ( ) const [inline]

Permet d'obtenir la valeur du ratio de hauteur de l'ellipse.

Renvoie

La valeur du ratio de hauteur de l'ellipse.

Définition à la ligne 187 du fichier ellipse.hpp.

Références yRadius.

```
188 {
189     return this->yRadius;
190 }
```

7.4.3.10 bool Ellipse : :operator != (const Ellipse &) const

Permet de savoir si deux Ellipse sont différentes.

Renvoie

`true` Si deux Ellipses sont différentes.

7.4.3.11 bool Ellipse : :operator== (const Ellipse &) const

Permet de savoir si deux Ellipse sont les mêmes.

Renvoie

`true` Si deux Ellipses sont identiques.

7.4.4 Documentation des données membres

7.4.4.1 Point Ellipse : :center [protected]

La position du centre de l'ellipse.

Définition à la ligne 23 du fichier ellipse.hpp.

Référencé par getCenter(), et Nuke : :getLocation().

7.4.4.2 double Ellipse : :height [protected]

La hauteur du rectangle circonscrit à l'ellipse.

Définition à la ligne 28 du fichier ellipse.hpp.

Référencé par getHeight().

7.4.4.3 double Ellipse : :width [protected]

La largeur du rectangle circonscrit à l'ellipse.

Définition à la ligne 33 du fichier ellipse.hpp.

Référencé par getWidth().

7.4.4.4 double Ellipse : :xRadius [protected]

La valeur de la demi hauteur au carré.

Définition à la ligne 38 du fichier ellipse.hpp.

Référencé par getXRadius().

7.4.4.5 double Ellipse : :yRadius [protected]

La valeur de la demi largeur au carré.

Définition à la ligne 43 du fichier ellipse.hpp.

Référencé par getYRadius().

La documentation de cette classe a été générée à partir du fichier suivant :
– model/geometry/ellipse.hpp

7.5 Référence de la classe Lens

Cette classe modélise les lentilles utilisées dans le jeu.

`#include <lens.hpp>`

Fonctions membres publiques

– Lens (const Point &, const int, const int, const int, const int)
  Créer une nouvelle lentille pouvant être un obstacle à un rayon : si le rayon souhaite passer au travers, il devra être d'une longueur d'onde comprise dans l'intervalle souhaité par cette lentille.
– const Point & getPosition () const
  Retourne la position du coin supérieur gauche du rectangle circonscrit à la lentille.
– int getMinWaveLength () const
  Retourne la longueur d'onde minimale des rayons autorisés à franchir la lentille.
– int getMaxWaveLength () const
  Retourne la longueur d'onde maximale des rayons autorisés à franchir la lentille.
– void reactToRay (Ray)
  Cette méthode est lancé lorsque la lentille courante est exposée à un rayon.
– Point ∗ includeRay (const Ray &) const
  Renseigne si la lentille est dans la trajectoire du rayon.
– bool operator== (const Lens &) const
  Permet de savoir si deux lentilles sont les mêmes.
– bool operator != (const Lens &) const
  Permet de savoir si deux lentilles sont différentes.

Attributs privés

– const Point upLeftCorner
  upLeftCorner
– const int wlMin
  wlMin
– const int wlMax
  wlMax

Membres hérités additionnels

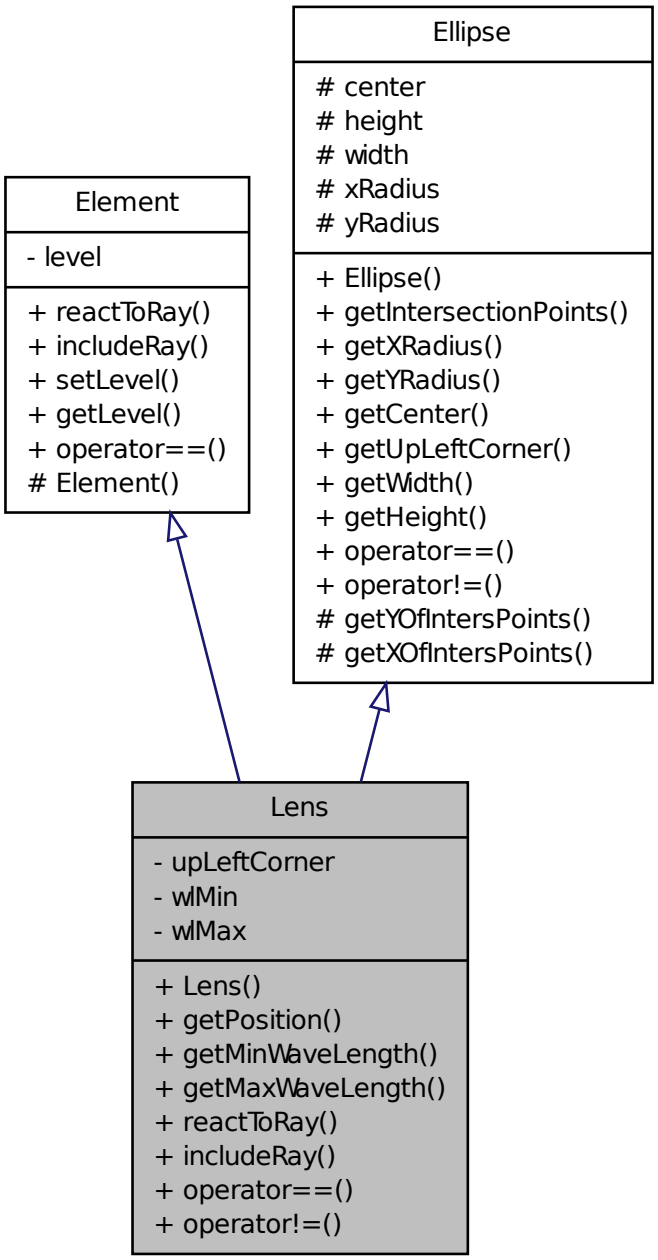
7.5.1 Description détaillée

Cette classe modélise les lentilles utilisées dans le jeu.

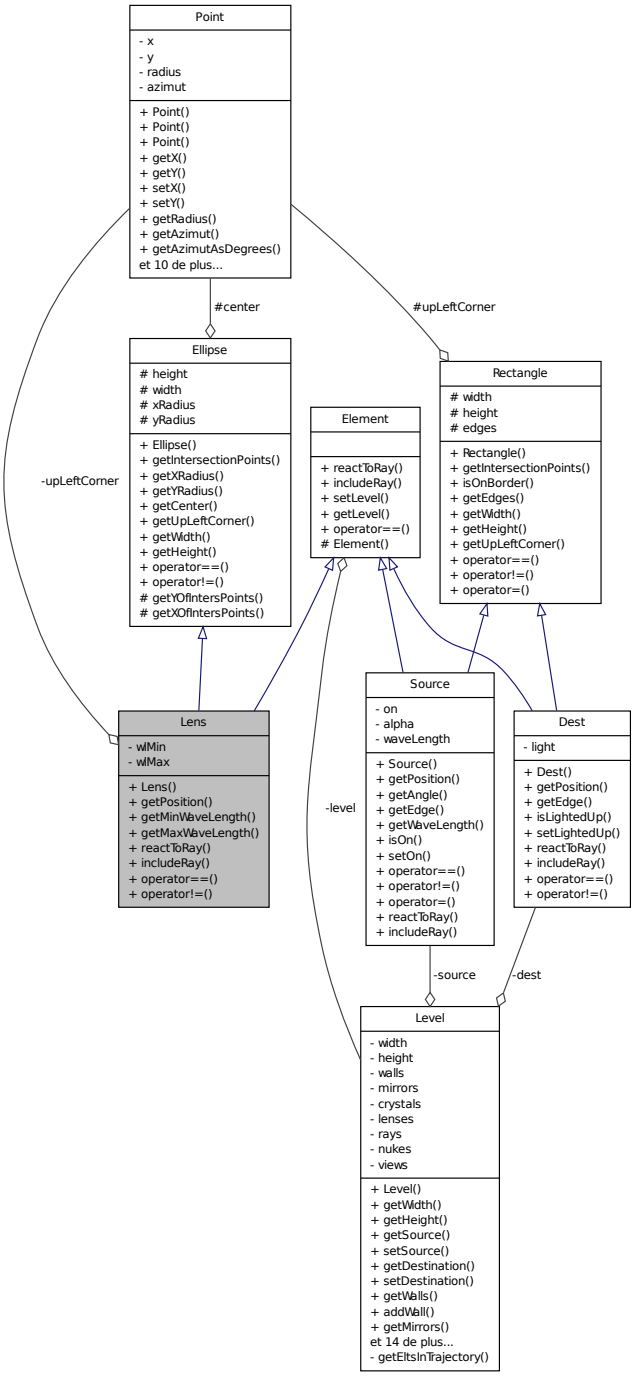
Une lentille est un objet rectangulaire qui ne laisse passer les rayons lumineux que dans un certain intervalle de longueur d'onde. Si un rayon lumineux se trouve dans l'intervalle de longueur d'onde autorisé, il traverse la lentille sans subir aucune modification. Sinon, la lentille se comporte comme un mur.

Définition à la ligne 22 du fichier lens.hpp.

Graphe d'héritage de Lens :



Graphe de collaboration de Lens :



7.5.2 Documentation des constructeurs et destructeur

7.5.2.1 Lens : :Lens (const Point & , const int , const int , const int , const int)

Créer une nouvelle lentille pouvant être un obstacle à un rayon : si le rayon souhaite passer au travers, il devra être d'une longueur d'onde comprise dans l'intervalle souhaité par cette lentille.

Dans le cas contraire, le rayon ne passera pas.

Paramètres

position	La position du coin supérieur gauche du rectangle circonscrit à l'ellipse modélisant la lentille.
width	La largeur du rectangle circonscrit à la lentille.
height	la hauteur du rectangle circonscrit à la lentille.
wlMin	La longueur d'onde minimale des rayons autorisés à franchir la lentille.
wlMax	La longueur d'onde maximale des rayons autorisés à franchir la lentille.

7.5.3 Documentation des fonctions membres

7.5.3.1 int Lens : :getMaxWaveLength ( ) const [inline]

Retourne la longueur d'onde maximale des rayons autorisés à franchir la lentille.

Renvoie

La longueur d'onde maximale des rayon autorisés à franchir la lentille.

Définition à la ligne 141 du fichier lens.hpp.

Références wlMax.

```
142 {
143     return this->wlMax;
144 }
```

7.5.3.2 int Lens : :getMinWaveLength ( ) const [inline]

Retourne la longueur d'onde minimale des rayons autorisés à franchir la lentille.

Renvoie

La longueur d'onde minimale des rayons autorisés à franchir la lentille.

Définition à la ligne 136 du fichier lens.hpp.

Références wlMin.

```
137 {
138     return this->wlMin;
139 }
```

7.5.3.3 const Point & Lens : :getPosition ( ) const [inline]

Retourne la position du coin supérieur gauche du rectangle circonscrit à la lentille.

Renvoie

La coordonnée cartésienne du coin supérieur gauche du rectangle modélisant la lentille.

Définition à la ligne 131 du fichier lens.hpp.

Références upLeftCorner.

```
132 {
133     return this->upLeftCorner;
134 }
```

7.5.3.4 Point* Lens : :includeRay (const Ray &) const [virtual]

Renseigne si la lentille est dans la trajectoire du rayon.

Paramètres

ray	Le rayon.

Renvoie

`true` Si la lentille se trouve dans la trajectoire du rayon entré en paramètre.

Implémente Element.

7.5.3.5 bool Lens : :operator != (const Lens &) const

Permet de savoir si deux lentilles sont différentes.

Renvoie

`true` Si les deux lentilles sont différentes.

7.5.3.6 bool Lens : :operator== (const Lens &) const

Permet de savoir si deux lentilles sont les mêmes.

Renvoie

`true` Si les deux lentilles sont les même.

7.5.3.7 void Lens : :reactToRay (Ray) [virtual]

Cette méthode est lancé lorsque la lentille courante est exposée à un rayon.

Elle va communiquer au niveau la fin du rayon si il ne peut pas passer ou ne va rien faire si le rayon passe.

Paramètres

ray	Un rayon percutant la lentille.

Implémente Element.

7.5.4 Documentation des données membres

7.5.4.1 const Point Lens : :upLeftCorner [private]

upLeftCorner

Définition à la ligne 30 du fichier lens.hpp.

Référencé par getPosition().

7.5.4.2 const int Lens : :wlMax [private]

wlMax

Définition à la ligne 40 du fichier lens.hpp.

Référencé par getMaxWaveLength().

7.5.4.3 const int Lens : :wlMin [private]

wlMin

Définition à la ligne 35 du fichier lens.hpp.

Référencé par getMinWaveLength().

La documentation de cette classe a été générée à partir du fichier suivant :
– model/elements/lens.hpp

7.6 Référence de la classe Level

Modélise une carte telle qu'utilisée dans le jeu.

`#include <level.hpp>`

Fonctions membres publiques

– Level (const double, const double)
  Instancie une carte de largeur et hauteur donnée.
– int getWidth () const
  Permet d'obtenir la longueur du niveau.
– int getHeight () const
  Permet d'obtenir la hauteur du niveau.
– Source & getSource ()
  Retourne la source de la carte.
– void setSource (const Source &)
  Change la source de la carte.
– const Dest & getDestination () const
  Retourne la destination de la carte.
– void setDestination (const Dest &)
  Change la destination de la carte.
– const std : :vector< Wall > & getWalls () const
  Retourne l'ensemble des murs de la carte.
– void addWall (const Wall &)
  Permet d'ajouter un mur sur la carte.
– std : :vector< Mirror > & getMirrors ()
  Retourne l'ensemble des miroirs de la carte.
– void addMirror (Mirror)
  Permet d'ajouter un miroir sur la carte.
– const std : :vector< Crystal > & getCrystals () const
  Retourne l'ensemble des cristaux de la carte.
– void addCrystal (Crystal)
  Permet d'ajouter un cristal sur la carte.
– const std : :vector< Lens > & getLenses () const
  Retourne l'ensemble des lentilles de la carte.
– void addLens (Lens)
  Permet d'ajouter une lentille sur la carte.
– std : :vector< Ray > & getRays ()
  Retourne l'ensemble des rayons de la carte.
– void setRays (const std : :vector< Ray > &)
  Change l'ensemble des rayons de la carte.
– const std : :vector< Nuke > & getNukes () const
  Retourne l'ensemble des bombes de la carte.
– void addNuke (const Nuke &)
  Permet d'ajouter une bombe sur la carte.
– bool thereIsAnExplodedNuke () const
  Renseigne si une bombe a explosé.
– void computeRay (Ray)
  Permet de calculer un rayon à partir du rayon entré en paramètre.
– void computeRays ()
  Calcule les rayons lumineux de la carte.
– void addView (LevelView ∗)
  Permet d'abonner une nouvelle vue au modèle.
– void notifyViews ()

Permet de notifier les vues abonnées au niveau que son état a changé.

Fonctions membres privées

– std : :map< Point ∗, Element ∗ > getEltsInTrajectory (const Ray &ray)
Permet d'obtenir une map contenant les élément se trouvant sur la trajectoire du rayon, ayant pour clé ; le point d'intersection avec cet élément.

Attributs privés

– const double width
La largeur du niveau.
– const double height
La hauteur du niveau.
– Source source {Point{0, 0}, 10, 30., 400}
La source du niveau.
– Dest dest {Point{0, 0}, 5}
La destination du niveau.
– std : :vector< Wall > walls
L'ensemble des murs du niveau, qu'ils soient ceux qui le délimitent ou des murs supplémentaires ajoutés au niveau même.
– std : :vector< Mirror > mirrors
mirrors L'ensemble des miroirs présents dans le niveau.
– std : :vector< Crystal > crystals
crystals L'ensemble des cristaux présents dans le niveau.
– std : :vector< Lens > lenses
lenses L'ensemble des lentilles présentes dans le niveau.
– std : :vector< Ray > rays
rays L'ensemble des rayons créés dans le niveau quand la source est allumée.
– std : :vector< Nuke > nukes
nukes L'ensemble des bombes créées dans le niveau.
– std : :vector< LevelView ∗ > views
views L'ensemble des vues qui observent le niveau.

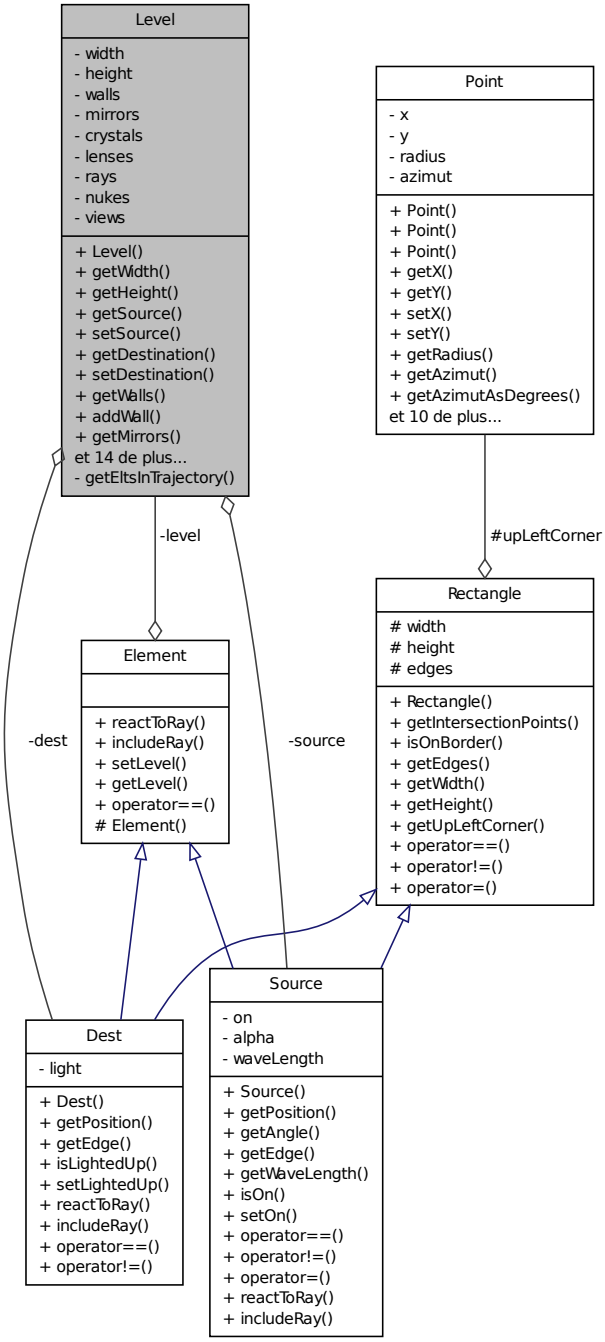
7.6.1 Description détaillée

Modélise une carte telle qu'utilisée dans le jeu.

Une carte est un ensemble de composant tels que des murs, des miroirs, etc.

Définition à la ligne 27 du fichier level.hpp.

Graphe de collaboration de Level :



7.6.2 Documentation des constructeurs et destructeur

7.6.2.1 Level : :Level (const double , const double)

Instancie une carte de largeur et hauteur donnée.

Quand une carte est crée, quatre murs dénotant ses bords sont automatiquement ajoutés à la carte.

La source et la destination sont initialisées à des valeurs par défaut inutilisables. Vous devez manuellement initialiser

la source et la destination via les fonctions appropriées.

Paramètres

w	la largeur de la carte
h	la hauteur de la carte

7.6.3 Documentation des fonctions membres

7.6.3.1 void Level : :addCrystal (Crystal)

Permet d'ajouter un cristal sur la carte.

Paramètres

newCrystal	nouveau cristal à ajouter.

7.6.3.2 void Level : :addLens (Lens)

Permet d'ajouter une lentille sur la carte.

Paramètres

newLens	nouvelle lentille à ajouter.

7.6.3.3 void Level : :addMirror (Mirror)

Permet d'ajouter un miroir sur la carte.

Paramètres

newMirror	nouveau miroir à ajouter.

7.6.3.4 void Level : :addNuke (const Nuke &)

Permet d'ajouter une bombe sur la carte.

Paramètres

newNuke	nouvelle bombe à ajouter.

7.6.3.5 void Level : :addView (LevelView *)

Permet d'abonner une nouvelle vue au modèle.

Paramètres

newView	Nouvelle vue abonnée au niveau.

7.6.3.6 void Level : :addWall (const Wall &)

Permet d'ajouter un mur sur la carte.

Paramètres

newWall	nouveau mur à ajouter.

7.6.3.7 void Level : :computeRay (Ray)

Permet de calculer un rayon à partir du rayon entré en paramètre.

Paramètres

ray	Rayon précèdent.

7.6.3.8 void Level : :computeRays ( )

Calcule les rayons lumineux de la carte.

7.6.3.9 const std : :vector< Crystal > & Level : :getCrystals ( ) const [inline]

Retourne l'ensemble des cristaux de la carte.

Renvoie

l'ensemble des cristaux de la carte

Définition à la ligne 308 du fichier level.hpp.

Références crystals.

```
309 {
310     return this->crystals;
311 }
```

7.6.3.10 const Dest & Level : :getDestination ( ) const [inline]

Retourne la destination de la carte.

Renvoie

la destination de la carte

Définition à la ligne 293 du fichier level.hpp.

Références dest.

```
294 {
295     return this->dest;
296 }
```

7.6.3.11 std : :map<Point *, Element *> Level : :getEltsInTrajectory (const Ray & *ray*) [private]

Permet d'obtenir une map contenant les élément se trouvant sur la trajectoire du rayon, ayant pour clé ; le point d'intersection avec cet élément.

Paramètres

ray	Rayon dont on désire obtenir les éléments sur sa trajectoire.

Renvoie

Une map contenant les élément se trouvant sur la trajectoire du rayon, ayant pour clé ; le point d'intersection avec cet élément.

7.6.3.12 int Level : :getHeight ( ) const [inline]

Permet d'obtenir la hauteur du niveau.

Renvoie

la hauteur du niveau.

Définition à la ligne 283 du fichier level.hpp.

Références height, et utilities : :round().

```
284 {
285     return std::round(this->height);
286 }
```

7.6.3.13 const std : :vector< Lens > & Level : :getLenses ( ) const [inline]

Retourne l'ensemble des lentilles de la carte.

Renvoie

l'ensemble des lentilles de la carte

Définition à la ligne 313 du fichier level.hpp.

Références lenses.

```
314 {
315     return this->lenses;
316 }
```

7.6.3.14 std : :vector< Mirror > & Level : :getMirrors ( ) [inline]

Retourne l'ensemble des miroirs de la carte.

Renvoie

l'ensemble des miroirs de la carte

Définition à la ligne 303 du fichier level.hpp.

Références mirrors.

```
304 {
305     return this->mirrors;
306 }
```

7.6.3.15 const std : :vector< Nuke > & Level : :getNukes ( ) const [inline]

Retourne l'ensemble des bombes de la carte.

Renvoie

l'ensemble des bombes de la carte

Définition à la ligne 323 du fichier level.hpp.

Références nukes.

```
324 {
325     return this->nukes;
326 }
```

7.6.3.16 std : :vector< Ray > & Level : :getRays ( ) [inline]

Retourne l'ensemble des rayons de la carte.

Renvoie

l'ensemble des rayons de la carte

Définition à la ligne 318 du fichier level.hpp.

Références rays.

```
319 {
320     return this->rays;
321 }
```

7.6.3.17 Source & Level : :getSource ( ) [inline]

Retourne la source de la carte.

Renvoie

la source de la carte.

Définition à la ligne 288 du fichier level.hpp.

Références source.

```
289 {
290     return this->source;
291 }
```

7.6.3.18 const std : :vector< Wall > & Level : :getWalls ( ) const [inline]

Retourne l'ensemble des murs de la carte.

Renvoie

l'ensemble des murs de la carte

Définition à la ligne 298 du fichier level.hpp.

Références walls.

```
299 {
300     return this->walls;
301 }
```

7.6.3.19 int Level : :getWidth ( ) const [inline]

Permet d'obtenir la longueur du niveau.

Renvoie

La longueur du niveau.

Définition à la ligne 278 du fichier level.hpp.

Références utilities : :round(), et width.

```
279 {
280     return std::round(this->width);
281 }
```

7.6.3.20 void Level : :notifyViews ( )

Permet de notifier les vues abonnées au niveau que son état a changé.

7.6.3.21 void Level : :setDestination (const Dest &)

Change la destination de la carte.

Paramètres

value	la destination de la carte

7.6.3.22 void Level : :setRays (const std : :vector< Ray > &)

Change l'ensemble des rayons de la carte.

Paramètres

le	nouvel ensemble de rayons de la carte

7.6.3.23 void Level : :setSource (const Source &)

Change la source de la carte.

Paramètres

value	la nouvelle source

7.6.3.24 bool Level : :thereIsAnExplodedNuke ( ) const

Renseigne si une bombe a explosé.

Renvoie

`true` Si une bombe a explosé.

7.6.4 Documentation des données membres

7.6.4.1 std : :vector<Crystal> Level : :crystals [private]

crystals L'ensemble des cristaux présents dans le niveau.

Définition à la ligne 65 du fichier level.hpp.

Référencé par getCrystals().

7.6.4.2 Dest Level : :dest {Point{0, 0}, 5} [private]

La destination du niveau.

Définition à la ligne 49 du fichier level.hpp.

Référencé par getDestination().

7.6.4.3 const double Level : :height [private]

La hauteur du niveau.

Définition à la ligne 39 du fichier level.hpp.

Référencé par getHeight().

7.6.4.4 std : :vector<Lens> Level : :lenses [private]

lenses L'ensemble des lentilles présentes dans le niveau.

Définition à la ligne 70 du fichier level.hpp.

Référencé par getLenses().

7.6.4.5 std : :vector<Mirror> Level : :mirrors [private]

mirrors L'ensemble des miroirs présents dans le niveau.

Définition à la ligne 60 du fichier level.hpp.

Référencé par getMirrors().

7.6.4.6 std : :vector<Nuke> Level : :nukes [private]

nukes L'ensemble des bombes créées dans le niveau.

Définition à la ligne 81 du fichier level.hpp.

Référencé par getNukes().

7.6.4.7 std : :vector<Ray> Level : :rays [private]

rays L'ensemble des rayons créés dans le niveau quand la source est allumée.

Définition à la ligne 76 du fichier level.hpp.

Référencé par getRays().

7.6.4.8 Source Level : :source {Point{0, 0}, 10, 30., 400} [private]

La source du niveau.

Définition à la ligne 44 du fichier level.hpp.

Référencé par getSource().

7.6.4.9 std : :vector<LevelView *> Level : :views [private]

views L'ensemble des vues qui observent le niveau.

Définition à la ligne 86 du fichier level.hpp.

7.6.4.10 std : :vector<Wall> Level : :walls [private]

L'ensemble des murs du niveau, qu'ils soient ceux qui le délimitent ou des murs supplémentaires ajoutés au niveau même.

Définition à la ligne 55 du fichier level.hpp.

Référencé par getWalls().

7.6.4.11 const double Level : :width [private]

La largeur du niveau.

Définition à la ligne 34 du fichier level.hpp.

Référencé par getWidth().

La documentation de cette classe a été générée à partir du fichier suivant :
– model/elements/level.hpp

7.7 Référence de la classe LevelView

Cette classe représente le niveau qui va être joué lors d'une partie.

`#include <levelview.hpp>`

Connecteurs publics

– void setLevelFilePath (const QString)
  Permet de changer le fichier de niveau et d'afficher ce niveau.
– void loadLevelFromFile ()
  Permet d'afficher le fichier niveau dont la vue courante contient le chemin en attribut.
– void updateDisplay ()
  Permet de rafraichir l'affichage lorsque le niveau change d'état.
– void displayEndOfGame ()
  Permet d'afficher une boite de dialogue informant l'utilisateur de la fin du jeu.

Signaux

– void displayingStarted ()
  Signale que le niveau doit être affiché.
– void displayingStopped ()
  Signale que le niveau ne doit plus être affiché.

Fonctions membres publiques

– LevelView (QWidget *parent=0)
  Permet de créer une vue du niveau.
– ~LevelView ()

Attributs privés

– QGraphicsScene * scene

La scène qui comportera l'ensemble des éléments graphiques de la partie.

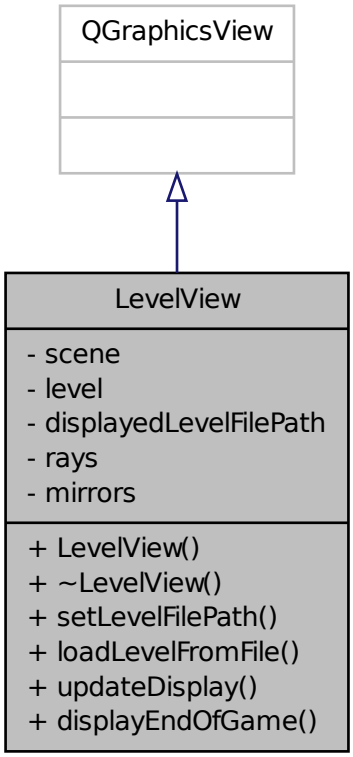
– Level * level
Le niveau quel la vue observe.
– std : :string displayedLevelFilePath
Le chemin du fichier chargé par l'utilisateur.
– std : :vector< QGraphicsLineItem * > rays
L'ensemble des rayons dessinés.
– std : :vector< MirrorView * > mirrors
L'ensemble des miroirs dessinés.

7.7.1 Description détaillée

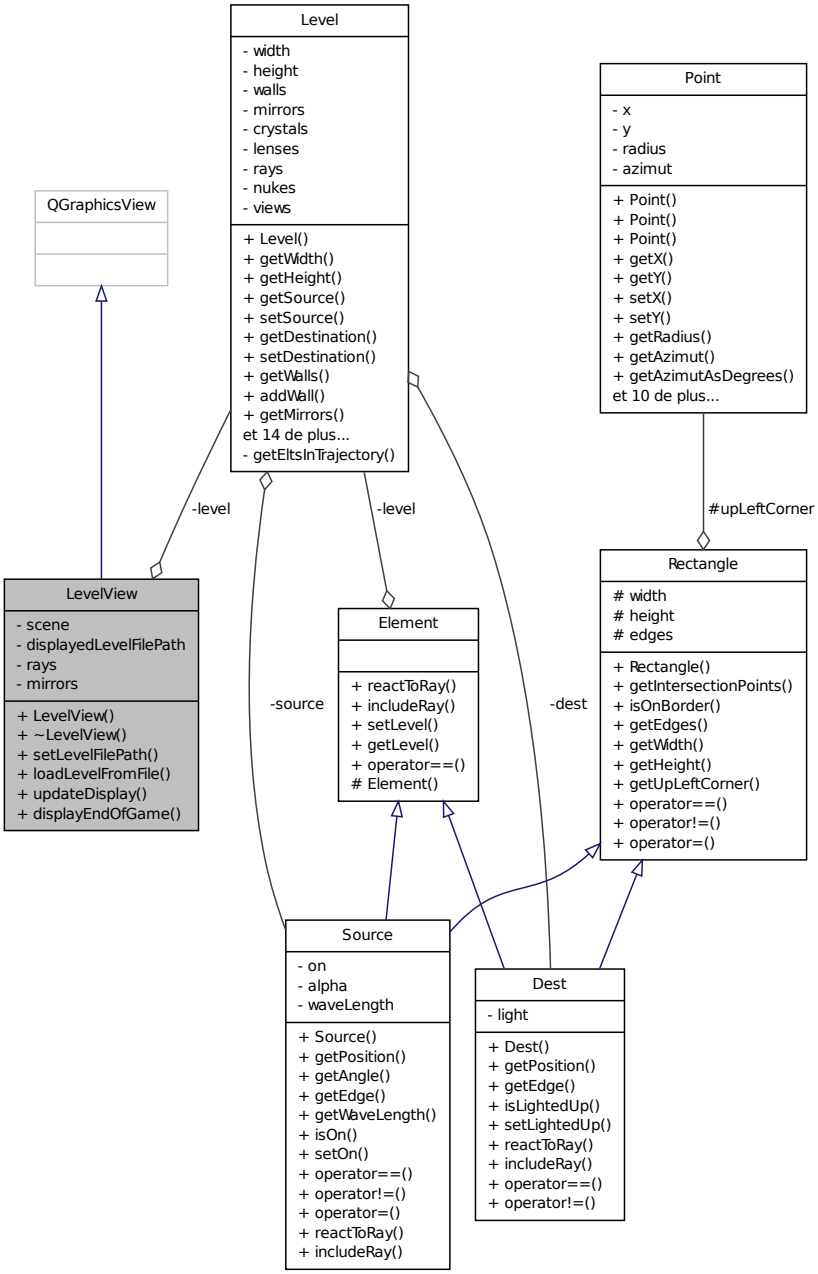
Cette classe représente le niveau qui va être joué lors d'une partie.

Définition à la ligne 18 du fichier levelview.hpp.

Graphe d'héritage de LevelView :



Graphe de collaboration de LevelView :



7.7.2 Documentation des constructeurs et destructeur

7.7.2.1 LevelView : :LevelView (QWidget ∗ *parent* = 0) [explicit]

Permet de créer une vue du niveau.

Paramètres

parent	L'objet graphique parent.

7.7.2.2 LevelView : :∼LevelView ( )

7.7.3 Documentation des fonctions membres

7.7.3.1 void LevelView : :displayEndOfGame ( ) [slot]

Permet d'afficher une boite de dialogue informant l'utilisateur de la fin du jeu.

7.7.3.2 void LevelView : :displayingStarted ( ) [signal]

Signale que le niveau doit être affiché.

7.7.3.3 void LevelView : :displayingStopped ( ) [signal]

Signale que le niveau ne doit plus être affiché.

7.7.3.4 void LevelView : :loadLevelFromFile ( ) [slot]

Permet d'afficher le fichier niveau dont la vue courante contient le chemin en attribut.

7.7.3.5 void LevelView : :setLevelFilePath (const QString) [slot]

Permet de changer le fichier de niveau et d'afficher ce niveau.

Paramètres

levelFile	chemin vers le fichier du nouveau niveau à afficher.

7.7.3.6 void LevelView : :updateDisplay ( ) [slot]

Permet de rafraichir l'affichage lorsque le niveau change d'état.

7.7.4 Documentation des données membres

7.7.4.1 std : :string LevelView : :displayedLevelFilePath [private]

Le chemin du fichier chargé par l'utilisateur.

Définition à la ligne 36 du fichier levelview.hpp.

7.7.4.2 Level∗ LevelView : :level [private]

Le niveau quel la vue observe.

Définition à la ligne 31 du fichier levelview.hpp.

7.7.4.3 std : :vector<MirrorView *> LevelView : :mirrors [private]

L'ensemble des miroirs dessinés.

Définition à la ligne 46 du fichier levelview.hpp.

7.7.4.4 std : :vector<QGraphicsLineItem *> LevelView : :rays [private]

L'ensemble des rayons dessinés.

Définition à la ligne 41 du fichier levelview.hpp.

7.7.4.5 QGraphicsScene* LevelView : :scene [private]

La scène qui comportera l'ensemble des éléments graphiques de la partie.

Définition à la ligne 26 du fichier levelview.hpp.

La documentation de cette classe a été générée à partir du fichier suivant :
– view/windows/levelview.hpp

7.8 Référence de la classe Line

Représente une droite sous la forme de son équation complète ; $eq \equiv y = slope \cdot x + indepTerm$.

`#include <line.hpp>`

Fonctions membres publiques

– Line (double, double, double=0)
Permet de construire une nouvelle droite initialisée.
– Line (const Point &, const Point &)
Permet de construire une droite à partir de deux points.
– Point * getIntersectionPoint (const Line &) const
Permet d'obtenir le point d'intersection entre la droite et celle entrée en paramètre.
– bool includes (const Point &) const
Renseigne si le point entré en paramètre est inclus dans la droite.
– double getSlope () const
Permet d'obtenir la pente de la droite.
– double getIndepTerm () const
Permet d'obtenir le terme indépendant.
– double getXValue () const
Permet de connaitre la valeur de x, cette valeur est incohérence si la droite n'est pas verticale.
– bool isVertical () const
Renseigne si la droite est verticale.
– double findX (const double) const
Permet de résoudre l'équation de la droite à partir d'une valeur de y entrée en paramètre.
– double findY (const double) const
Permet de résoudre l'équation de la droite à partir d'une valeur de x entrée en paramètre.
– Line & operator= (const Line &)
Permet de copier une ligne.
– bool operator== (const Line &) const
Permet de savoir si deux lignes sont identiques.
– bool operator != (const Line &) const
Permet de savoir si deux lignes sont différentes.

Attributs protégés

– double slope
slope Valeur du coefficient angulaire de la droite.
– double indepTerm

indepTerm Contient la valeur du terme indépendant de la droite.

– double xValue

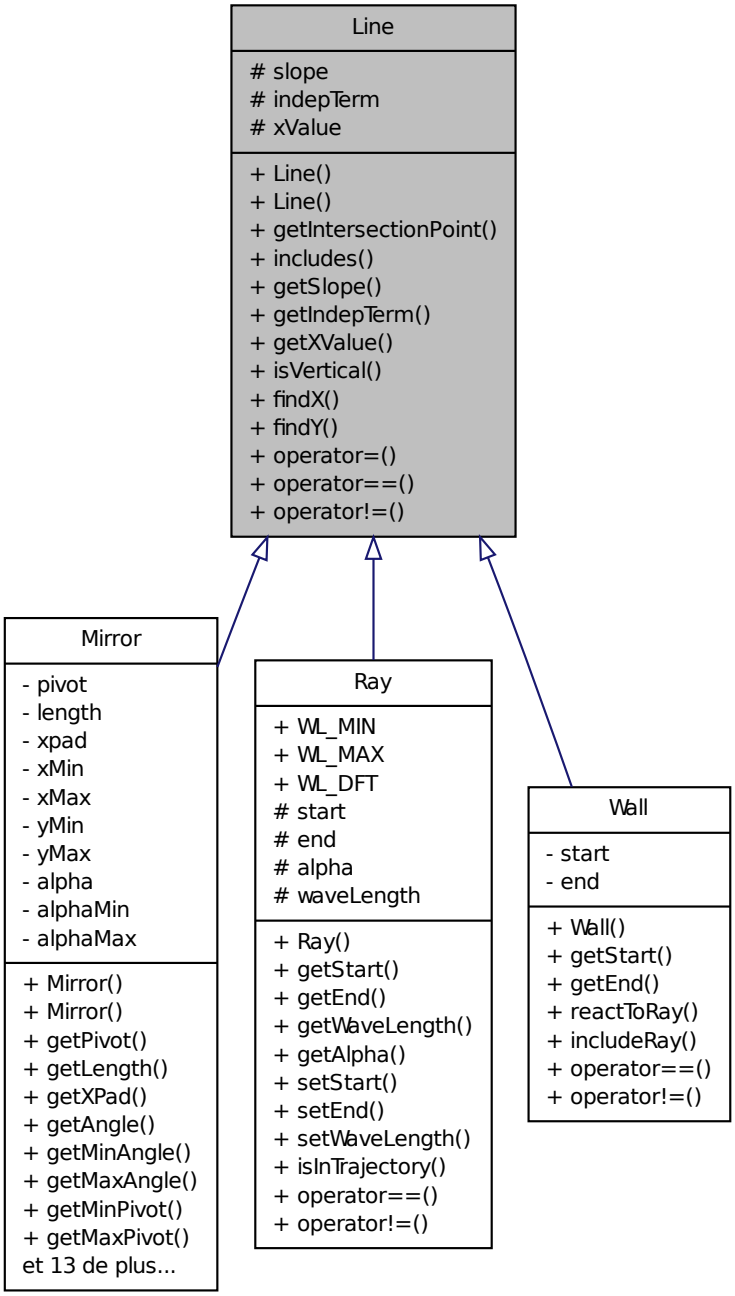
xValue Contient la valeur de x lorsque la droite est verticale.

7.8.1 Description détaillée

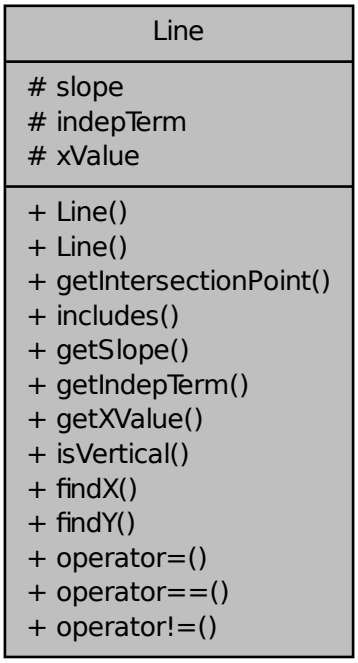
Représente une droite sous la forme de son équation complète ; $eq \equiv y = slope \cdot x + indepTerm$.

Définition à la ligne 13 du fichier line.hpp.

Graphe d'héritage de Line :



Graphe de collaboration de Line :



7.8.2 Documentation des constructeurs et destructeur

7.8.2.1 Line : :Line (double , double , double = 0)

Permet de construire une nouvelle droite initialisée.

Paramètres

slope	Pente de la droite.
indepTerm	Terme indépendant de la droite.
xValue	Valeur de x si la droite est verticale.

7.8.2.2 Line : :Line (const Point & , const Point &)

Permet de construire une droite à partir de deux points.

Paramètres

start	Point le plus près de l'origine au niveau de l'abscisse.
end	Point le plus éloigné de l'origine au niveau de l'abscisse.

7.8.3 Documentation des fonctions membres

7.8.3.1 double Line : :findX (const double) const

Permet de résoudre l'équation de la droite à partir d'une valeur de y entrée en paramètre.

Paramètres

y	Valeur de y pour résoudre l'équation de la droite.

Renvoie

le valeur de x après résolution de l'équation de la droite à partir d'une valeur de y entrée en paramètre.

7.8.3.2 double Line : :findY (const double) const

Permet de résoudre l'équation de la droite à partir d'une valeur de x entrée en paramètre.

Paramètres

x	Valeur de x pour résoudre l'équation de la droite.

Renvoie

la valeur de y après résolution de l'équation de la droite à partir d'une valeur de y entrée en paramètre.

7.8.3.3 double Line : :getIndepTerm ( ) const [inline]

Permet d'obtenir le terme indépendant.

Renvoie

Le terme indépendant.

Définition à la ligne 160 du fichier line.hpp.

Références indepTerm.

```
161 {
162     return this->indepTerm;
163 }
```

7.8.3.4 Point* Line : :getIntersectionPoint (const Line &) const

Permet d'obtenir le point d'intersection entre la droite et celle entrée en paramètre.

Paramètres

line	Droite dont on désire obtenir le point d'intersection avec la droite courante.

Renvoie

Un pointeur vers un le point d'intersection entre la droite et celle entrée en paramètre si il existe, un pointeur nul sinon.

7.8.3.5 double Line : :getSlope ( ) const [inline]

Permet d'obtenir la pente de la droite.

Renvoie

La pente de la droite.

Définition à la ligne 155 du fichier line.hpp.

Références slope.

```
156 {
157     return this->slope;
158 }
```

7.8.3.6 double Line : :getXValue ( ) const [inline]

Permet de connaitre la valeur de x, cette valeur est incohérence si la droite n'est pas verticale.

Renvoie

La valeur de x si la droite est verticale.

Définition à la ligne 165 du fichier line.hpp.

Références xValue.

```
166 {
167     return this->xValue;
168 }
```

7.8.3.7 bool Line : :includes (const Point &) const

Renseigne si le point entré en paramètre est inclus dans la droite.

Paramètres

point	Point dont on désire savoir s'il est inclus dans la droite.

Renvoie

`true` si le point entré en paramètre est inclus dans la droite.

7.8.3.8 bool Line : :isVertical ( ) const

Renseigne si la droite est verticale.

Renvoie

`true` Si la droite est verticale.

7.8.3.9 bool Line : :operator != (const Line &) const

Permet de savoir si deux lignes sont différentes.

Renvoie

`true` Si deux lignes sont différentes.

7.8.3.10 Line& Line : :operator= (const Line &)

Permet de copier une ligne.

Renvoie

La ligne courante représentant la ligne passée en paramètre.

7.8.3.11 bool Line : :operator== (const Line &) const

Permet de savoir si deux lignes sont identiques.

Renvoie

`true` Si deux lignes sont identiques.

7.8.4 Documentation des données membres

7.8.4.1 double Line : :indepTerm [protected]

indepTerm Contient la valeur du terme indépendant de la droite.

Définition à la ligne 26 du fichier line.hpp.

Référencé par getIndepTerm().

7.8.4.2 double Line : :slope [protected]

slope Valeur du coefficient angulaire de la droite.

Définition à la ligne 21 du fichier line.hpp.

Référencé par getSlope().

7.8.4.3 double Line : :xValue [protected]

xValue Contient la valeur de x lorsque la droite est verticale.

Définition à la ligne 31 du fichier line.hpp.

Référencé par getXValue().

La documentation de cette classe a été générée à partir du fichier suivant :
– model/geometry/line.hpp

7.9 Référence de la classe MainMenu

Cette classe représente le menu principal du jeu permettant de.

`#include <mainmenu.hpp>`

Connecteurs publics

– void selectNewLevelFile ()
Permet de faire sélectionner un fichier de niveau par l'utilisateur.
– void displayRules ()
Permet d'afficher une fenêtre de dialogue contenant les règles et les commandes du jeu.

Signaux

– void newLevelFileSelected (const QString)
Signale qu'un nouveau fichier a été sélectionné par l'utilisateur.

Fonctions membres publiques

– MainMenu (QWidget ∗=0)
Permet de créer un menu du jeu.
– ∼MainMenu ()

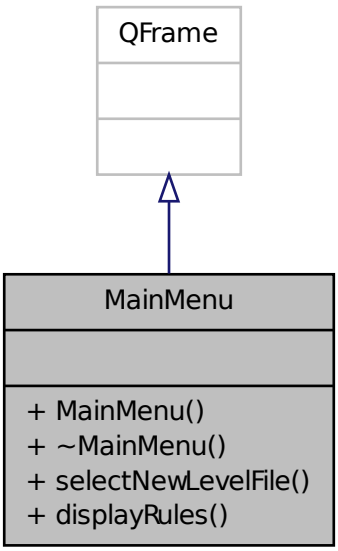
7.9.1 Description détaillée

Cette classe représente le menu principal du jeu permettant de.

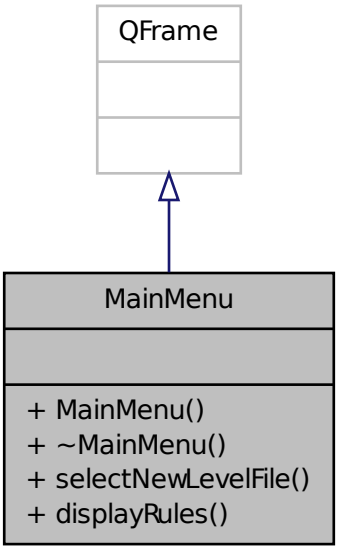
– sélectionner un niveau à jouer,
– lire les règles du jeu,
– quitter le jeu.

Définition à la ligne 14 du fichier mainmenu.hpp.

Graphe d'héritage de MainMenu :



Graphe de collaboration de MainMenu :



7.9.2 Documentation des constructeurs et destructeur

7.9.2.1 MainMenu : :MainMenu (QWidget * = 0) [explicit]

Permet de créer un menu du jeu.

7.9.2.2 MainMenu : :~MainMenu ( )

7.9.3 Documentation des fonctions membres

7.9.3.1 void MainMenu : :displayRules ( ) [slot]

Permet d'afficher une fenêtre de dialogue contenant les règles et les commandes du jeu.

7.9.3.2 void MainMenu : :newLevelFileSelected (const QString) [signal]

Signale qu'un nouveau fichier a été sélectionné par l'utilisateur.

Paramètres

newLevelFile	Chemin vers le nouveau fichier sélectionné.

7.9.3.3 void MainMenu : :selectNewLevelFile ( ) [slot]

Permet de faire sélectionner un fichier de niveau par l'utilisateur.

Paramètres

newLevelFile	Chemin vers le nouveau fichier sélectionné.

La documentation de cette classe a été générée à partir du fichier suivant :
– view/windows/mainmenu.hpp

7.10 Référence de la classe MainWindow

Cette classe est la fenêtre principale du jeu qui englobe toutes les autres vues.

`#include <mainwindow.hpp>`

Connecteurs publics

– void displayMainMenu ()
Permet d'afficher le menu principal du jeu.
– void displayLevel ()
Permet d'afficher le niveau.

Fonctions membres publiques

– MainWindow (QWidget ∗=0)
Créer une fenêtre principale du jeu.
– ∼MainWindow ()

Fonctions membres privées

– void setMenuBar ()
Configure la bar de menu.
– void connectAll ()
Créer toutes les connections SLOT / SIGNAL.

Attributs privés

– MainMenu ∗ mainMenu
Le menu de sélection de niveau du jeu.
– LevelView ∗ levelView
La vue de la partie qui est lancée.
– QMenuBar ∗ bar
La bar de menu du jeu.
– QMenu ∗ menu
Le menu principal du jeu.

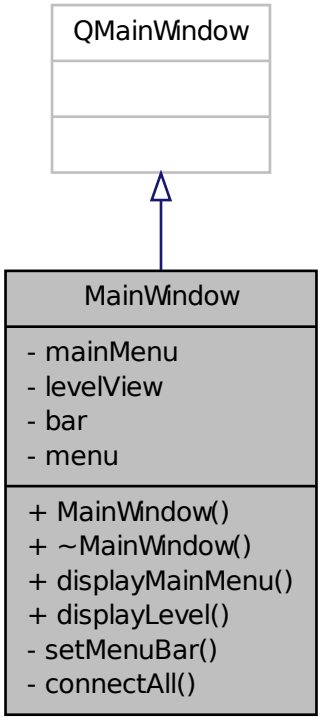
7.10.1 Description détaillée

Cette classe est la fenêtre principale du jeu qui englobe toutes les autres vues.

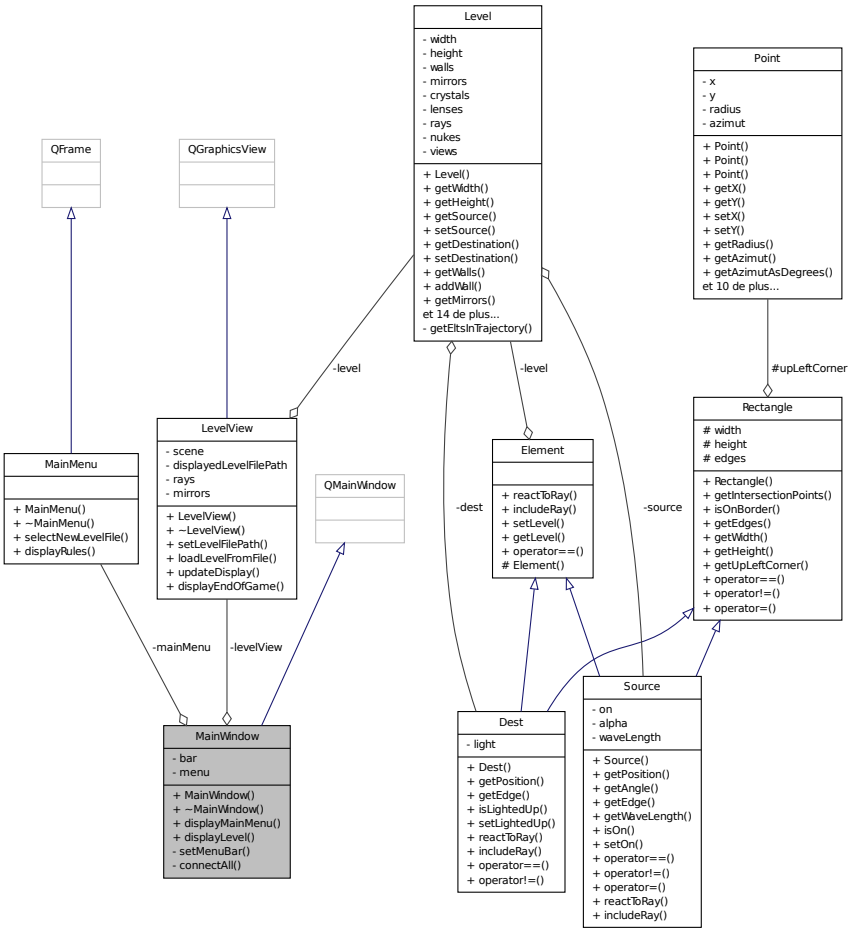
Elle permet, notamment, d'avoir un menu.

Définition à la ligne 15 du fichier mainwindow.hpp.

Graphe d'héritage de MainWindow :



Graphe de collaboration de MainWindow :



7.10.2 Documentation des constructeurs et destructeur

7.10.2.1 MainWindow : :MainWindow (QWidget * = 0) [explicit]

Créer une fenêtre principale du jeu.

7.10.2.2 MainWindow : :~MainWindow ( )

7.10.3 Documentation des fonctions membres

7.10.3.1 void MainWindow : :connectAll ( ) [private]

Créer toutes les connections SLOT / SIGNAL.

7.10.3.2 void MainWindow : :displayLevel ( ) [slot]

Permet d'afficher le niveau.

7.10.3.3 void MainWindow : :displayMainMenu ( ) [slot]

Permet d'afficher le menu principal du jeu.

7.10.3.4 void MainWindow : :setMenuBar ( ) [private]

Configure la bar de menu.

7.10.4 Documentation des données membres

7.10.4.1 QMenuBar* MainWindow : :bar [private]

La bar de menu du jeu.

Définition à la ligne 34 du fichier mainwindow.hpp.

7.10.4.2 LevelView* MainWindow : :levelView [private]

La vue de la partie qui est lancée.

Définition à la ligne 29 du fichier mainwindow.hpp.

7.10.4.3 MainMenu* MainWindow : :mainMenu [private]

Le menu de sélection de niveau du jeu.

Définition à la ligne 24 du fichier mainwindow.hpp.

7.10.4.4 QMenu* MainWindow : :menu [private]

Le menu principal du jeu.

Définition à la ligne 39 du fichier mainwindow.hpp.

La documentation de cette classe a été générée à partir du fichier suivant :
– view/windows/mainwindow.hpp

7.11 Référence de la classe Mirror

Cette classe modélise les miroirs utilisés dans le jeu.

`#include <mirror.hpp>`

Fonctions membres publiques

– Mirror (const Point &, int, int, double)
Instancie un miroir en une position donnée, d'une certaine longueur et orienté d'un certain angle.
– Mirror (const Point &, int, int, double, Point, Point, double, double)
Instancie un miroir en une position donnée, d'une certaine longueur et orienté d'un certain angle.
– const Point & getPivot () const
Retourne la position du pivot du miroir.
– int getLength () const
Retourne la longueur du miroir.
– int getXPad () const
Retourne le décalage du pivot par rapport au bord gauche du miroir.
– double getAngle () const
Retourne l'inclinaison du miroir.

- double getMinAngle () const
  Retourne l'inclinaison minimum du miroir.
- double getMaxAngle () const
  Retourne l'inclinaison minimum du miroir.
- Point getMinPivot () const
  Retourne la position minimum du miroir.
- Point getMaxPivot () const
  Retourne la position maximum du miroir.
- bool setPivot (const Point &)
  Déplace le miroir en la position donnée, si celle-ci est autorisée.
- bool setAngle (double)
  Pivote le miroir sur un angle donné, si celui-ci est autorisé.
- bool rotate (double)
  *Permet d'**essayer** d'effectuer une rotation du miroir courant.*
- bool translate (const double, const double)
  Permet de déplacer le miroir dans le plan en lui donnant un abscisse et une ordonnée de translation.
- Point getStart () const
  Retourne le point de départ du segment de droite représentant le miroir.
- Point getEnd () const
  Retourne le point d' arrivé du segment de droite représentant le miroir.
- void getBounds (Point *, Point *) const
  Permet d'obtenir le point de départ et d'arrivé du segment de droite représentant le miroir ; pour éviter de retourner un conteneur de points, ceux-ci sont passés en entrée sortie des paramètres.
- bool checkAngleRange (double) const
  Retoune vrai si le miroir peut être pivoté sur l'angle donné, retourne faux sinon.
- bool checkPivotRange (const Point &) const
  Retoune vrai si le miroir peut être déplacé en la position donnée, retourne faux sinon.
- void reactToRay (Ray)
  Réaction à l'exposition d'un rayon ; celui-ci est réfléchi selon le principe naturel de la réflexion de la lumière dans l'air.
- Point * includeRay (const Ray &) const
  Renseigne si le miroir est dans la trajectoire du rayon.
- bool operator== (const Mirror &) const
  Permet de savoir si deux miroirs sont les même.
- bool operator != (const Mirror &) const
  Permet de savoir si deux miroirs sont différents.

Attributs privés

- Point pivot
  Le point de pivot du miroir autour du quel celui-ci peut effectuer une rotation.
- int length
  La longueur totale du miroir.
- int xpad
  La longueur séparant le point de pivot d'un point extrême du miroir.
- double xMin
  Limite minimale d'abscisse où peut se situer le pivot du miroir.
- double xMax
  Limite maximale d'abscisse où peut se situer le pivot du miroir.
- double yMin
  Limite minimale d'ordonnée où peut se situer le pivot du miroir.
- double yMax
  Limite maximale d'ordonnée où peut se situer le pivot du miroir.
- double alpha
  L'angle actuel de rotation du miroir.
- double alphaMin
  L'angle minimum dans lequel peut se trouver le miroir.
- double alphaMax
  L'angle maximum dans lequel peut se trouver le miroir.

Membres hérités additionnels

7.11.1 Description détaillée

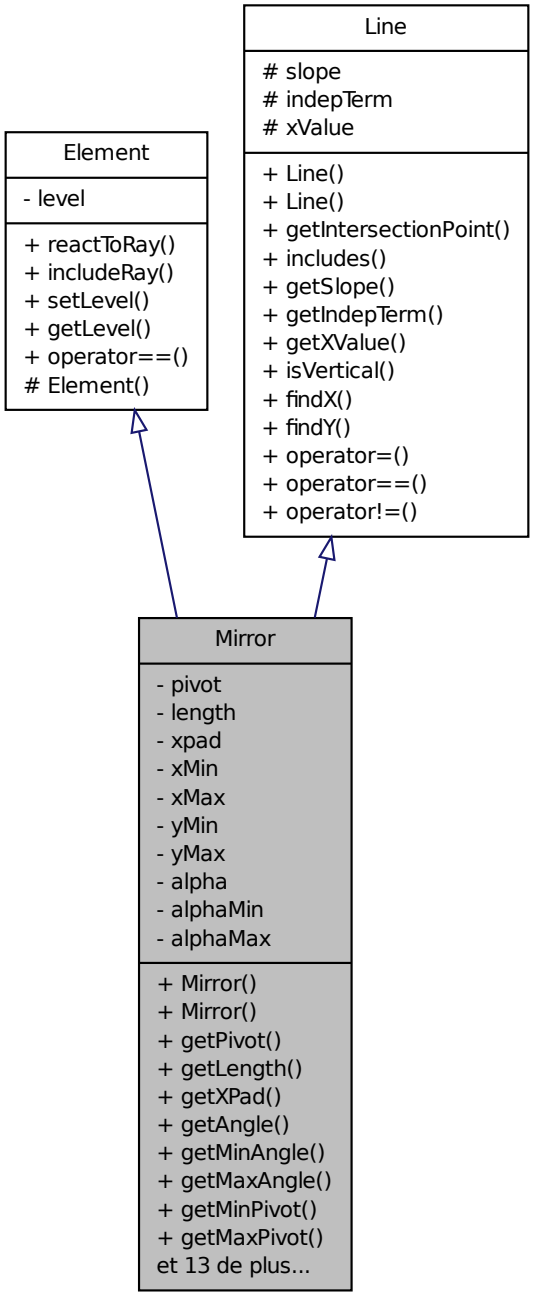
Cette classe modélise les miroirs utilisés dans le jeu.

Un miroir est un segment de droite dont la propriété est de réfléchir la lumière d'un seul côté uniquement. Si un

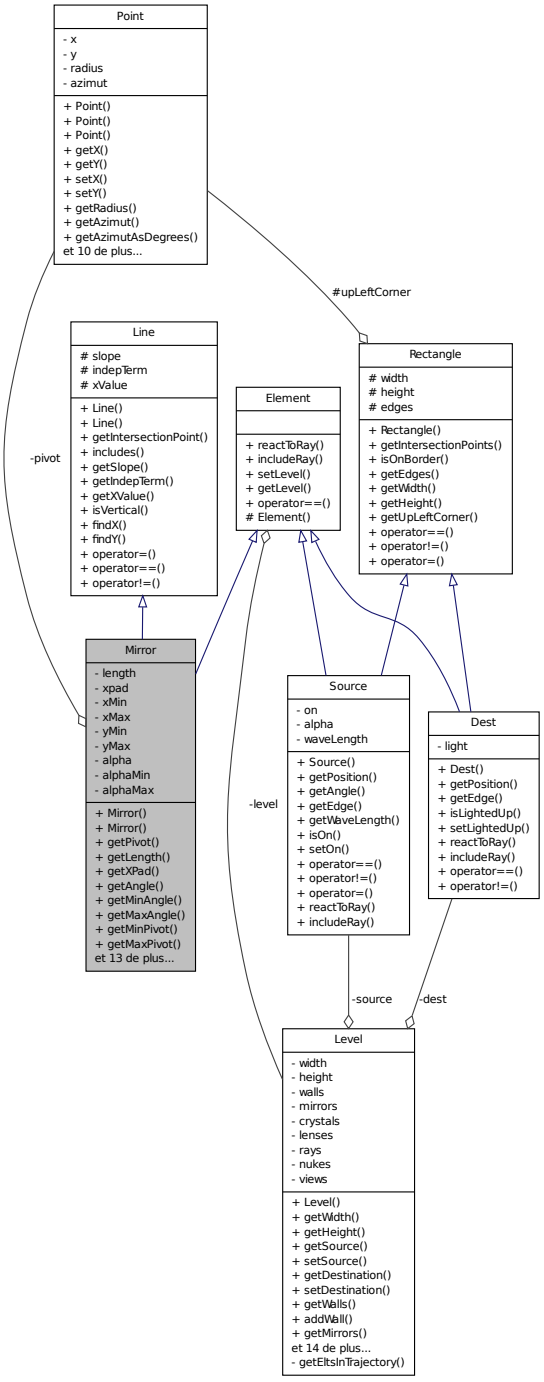
rayon lumineux touche un miroir du côté non réfléchissant, le miroir se comporte comme un mur. Les miroirs sont capables d'être déplacés et pivotés dans une certaine limite.

Définition à la ligne 21 du fichier mirror.hpp.

Graphe d'héritage de Mirror :



Graphe de collaboration de Mirror :



7.11.2 Documentation des constructeurs et destructeur

7.11.2.1 Mirror : :Mirror (const Point & , int , int , double)

Instancie un miroir en une position donnée, d'une certaine longueur et orienté d'un certain angle.

Comme dans ce constructeur les limites de déplacement et de rotation du miroir ne sont pas définies, ce miroir peut se déplacer et pivoter librement.

Paramètres

pivot	La position du pivot du miroir.
xpad	Le décalage du pivot par rapport au bord gauche du miroir.
length	La longueur du miroir.
alpha	L'angle d'inclinaison du miroir.

7.11.2.2 Mirror : :Mirror (const Point & , int , int , double , Point , Point , double , double)

Instancie un miroir en une position donnée, d'une certaine longueur et orienté d'un certain angle.

Ce constructeur permet également aux miroirs de pivoter dans une certaine limite. Si l'intervalle de limite de déplacement (e.g., sur les abscisses) [a,b] est tel que si :
– a = b, le miroir ne peut être déplacé sur l'axe considéré
– a $<$ b, le miroir pivote dans le sens horloger
– a = b le miroir ne peut pas pivoter
– a $>$ b, le miroir pivote dans le sens anti-horloger

Paramètres

pivot	La position du pivot du miroir.
xpad	Le décalage du pivot par rapport au bord gauche du miroir.
length	La longueur du miroir.
alpha	L'angle d'inclinaison du miroir.
pointMin	Le point de coordonnées minimum.
pointMax	Le point de coordonnées maximum.
alphaMin	L'angle d'inclinaison minimum du miroir (en radian).
alphaMax	L'angle d'inclinaison maximum du miroir (en radian).

7.11.3 Documentation des fonctions membres

7.11.3.1 bool Mirror : :checkAngleRange (double) const

Retoune vrai si le miroir peut être pivoté sur l'angle donné, retourne faux sinon.

Renvoie

`true` si le miroir peut être pivoté sur l'angle donné, retourne faux sinon.

7.11.3.2 bool Mirror : :checkPivotRange (const Point &) const

Retoune vrai si le miroir peut être déplacé en la position donnée, retourne faux sinon.

Renvoie

`true` si le miroir peut être déplacé en la position donnée, retourne faux sinon.

7.11.3.3 double Mirror : :getAngle ( ) const [inline]

Retourne l'inclinaison du miroir.

Renvoie

l'inclinaison du miroir.

Définition à la ligne 338 du fichier mirror.hpp.

Références alpha.

```
339 {
340     return this->alpha;
341 }
```

7.11.3.4 void Mirror : :getBounds (Point ∗ , Point ∗) const

Permet d'obtenir le point de départ et d'arrivé du segment de droite représentant le miroir ; pour éviter de retourner un conteneur de points, ceux-ci sont passés en entrée sortie des paramètres.

7.11.3.5 Point Mirror : :getEnd ( ) const

Retourne le point d' arrivé du segment de droite représentant le miroir.

Renvoie

Le point d'arrivé du segment de droite représentant le miroir.

7.11.3.6 int Mirror : :getLength ( ) const [inline]

Retourne la longueur du miroir.

Renvoie

La longueur du miroir.

Définition à la ligne 328 du fichier mirror.hpp.

Références length.

```
329 {
330     return this->length;
331 }
```

7.11.3.7 double Mirror : :getMaxAngle ( ) const [inline]

Retourne l'inclinaison minimum du miroir.

Si l'intervalle de limite d'inclinaison [a,b] est tel que :
– a $<$ b, le miroir pivote dans le sens horloger
– a = b, le miroir ne peut pas pivoter
– a $>$ b, le miroir pivote dans le sens anti-horloger
– Si a = b = 0, le miroir peut être pivoté librement

Renvoie

l'inclinaison maximum du miroir en radian.

Définition à la ligne 348 du fichier mirror.hpp.

Références alphaMax.

```
349 {
350     return this->alphaMax;
351 }
```

7.11.3.8 Point Mirror : :getMaxPivot ( ) const

Retourne la position maximum du miroir.

Si l'intervalle de limite de déplacement (e.g., sur les abscisses) [a,b] est tel que a = b, le miroir ne peut être déplacé sur l'axe considéré. Si a = b = 0, le miroir peut être déplacé librement.

Renvoie

la position minimum du miroir.

7.11.3.9 double Mirror : :getMinAngle ( ) const [inline]

Retourne l'inclinaison minimum du miroir.

Si l'intervalle de limite d'inclinaison [a,b] est tel que :
- a < b, le miroir pivote dans le sens horloger
- a = b, le miroir ne peut pas pivoter
- a > b, le miroir pivote dans le sens anti-horloger
- Si a = b = 0, le miroir peut être pivoté librement

Renvoie

l'inclinaison minimum du miroir en radian.

Définition à la ligne 343 du fichier mirror.hpp.

Références alphaMin.

```
344 {
345     return this->alphaMin;
346 }
```

7.11.3.10 Point Mirror : :getMinPivot ( ) const

Retourne la position minimum du miroir.

Si l'intervalle de limite de déplacement (e.g., sur les abscisses) [a,b] est tel que :
- a = b, le miroir ne peut être déplacé sur l'axe considéré
- a = b = 0, le miroir peut être déplacé librement

Renvoie

la position minimum du miroir.

7.11.3.11 const Point & Mirror : :getPivot ( ) const [inline]

Retourne la position du pivot du miroir.

Renvoie

La position du pivot du miroir.

Définition à la ligne 323 du fichier mirror.hpp.

Références pivot.

```
324 {
325     return this->pivot;
326 }
```

7.11.3.12 Point Mirror : :getStart ( ) const

Retourne le point de départ du segment de droite représentant le miroir.

Renvoie

Le point de départ du segment de droite représentant le miroir.

7.11.3.13 int Mirror : :getXPad ( ) const [inline]

Retourne le décalage du pivot par rapport au bord gauche du miroir.

Renvoie

Le décalage du pivot par rapport au bord gauche du miroir.

Définition à la ligne 333 du fichier mirror.hpp.

Références xpad.

```
334 {
335     return this->xpad;
336 }
```

7.11.3.14 Point* Mirror : :includeRay (const Ray &) const [virtual]

Renseigne si le miroir est dans la trajectoire du rayon.

Paramètres

ray	Le rayon.

Renvoie

`true` Si le miroir se trouve dans la trajectoire du rayon entré en paramètre.

Implémente Element.

7.11.3.15 bool Mirror : :operator != (const Mirror &) const

Permet de savoir si deux miroirs sont différents.

Renvoie

`true` Si deux miroirs sont différents.

7.11.3.16 bool Mirror : :operator== (const Mirror &) const

Permet de savoir si deux miroirs sont les même.

Renvoie

`true` Si deux miroirs sont les même.

7.11.3.17 void Mirror : :reactToRay (Ray) [virtual]

Réaction à l'exposition d'un rayon ; celui-ci est réfléchi selon le principe naturel de la réflexion de la lumière dans l'air.

Cette méthode communiquera au niveau de prendre en compte le nouveau rayon créé.

Paramètres

ray	Le rayon incident.

Implémente Element.

7.11.3.18 bool Mirror : :rotate (double)

Permet d'**essayer** d'effectuer une rotation du miroir courant.

Paramètres

alpha	L'angle de rotation en degrés.

Renvoie

`true` Si le miroir ne sort pas des limites après rotation.

7.11.3.19 bool Mirror : :setAngle (double)

Pivote le miroir sur un angle donné, si celui-ci est autorisé.

Voir également

Mirror : :getAngle()

Renvoie

`true` Si la rotation a été effectuée.

7.11.3.20 bool Mirror : :setPivot (const Point &)

Déplace le miroir en la position donnée, si celle-ci est autorisée.

Voir également

Mirror : :getPivot()

Renvoie

`true` Si le déplacement a été effectué.

7.11.3.21 bool Mirror : :translate (const double , const double)

Permet de déplacer le miroir dans le plan en lui donnant un abscisse et une ordonnée de translation.

Paramètres

x	Le déplacement sur l'axe des abscisses.
y	Le déplacement sur l'axe des ordonnées.

Renvoie

`true` Si le miroir ne sort pas des limites après translations.

7.11.4 Documentation des données membres

7.11.4.1 double Mirror : :alpha [private]

L'angle actuel de rotation du miroir.

Définition à la ligne 62 du fichier mirror.hpp.

Référencé par getAngle().

7.11.4.2 double Mirror : :alphaMax [private]

L'angle maximum dans lequel peut se trouver le miroir.

Définition à la ligne 72 du fichier mirror.hpp.

Référencé par getMaxAngle().

7.11.4.3 double Mirror : :alphaMin [private]

L'angle minimum dans lequel peut se trouver le miroir.

Définition à la ligne 67 du fichier mirror.hpp.

Référencé par getMinAngle().

7.11.4.4 int Mirror : :length [private]

La longueur totale du miroir.

Définition à la ligne 32 du fichier mirror.hpp.

Référencé par getLength().

7.11.4.5 Point Mirror : :pivot [private]

Le point de pivot du miroir autour du quel celui-ci peut effectuer une rotation.

Définition à la ligne 27 du fichier mirror.hpp.

Référencé par getPivot().

7.11.4.6 double Mirror : :xMax [private]

Limite maximale d'abscisse où peut se situer le pivot du miroir.

Définition à la ligne 47 du fichier mirror.hpp.

7.11.4.7 double Mirror : :xMin [private]

Limite minimale d'abscisse où peut se situer le pivot du miroir.

Définition à la ligne 42 du fichier mirror.hpp.

7.11.4.8 int Mirror : :xpad [private]

La longueur séparant le point de pivot d'un point extrême du miroir.

Définition à la ligne 37 du fichier mirror.hpp.

Référencé par getXPad().

7.11.4.9 double Mirror : :yMax [private]

Limite maximale d'ordonnée où peut se situer le pivot du miroir.

Définition à la ligne 57 du fichier mirror.hpp.

7.11.4.10 double Mirror : :yMin [private]

Limite minimale d'ordonnée où peut se situer le pivot du miroir.

Définition à la ligne 52 du fichier mirror.hpp.

La documentation de cette classe a été générée à partir du fichier suivant :
– model/elements/mirror.hpp

7.12 Référence de la classe MirrorView

Cette classe représente graphiquement un miroir du jeu permettant d'interagir avec lui à l'aide de la souris et du clavier.

`#include <mirrorview.hpp>`

Fonctions membres publiques

– MirrorView (Mirror ∗mirror)
Construit une vue de miroir lié à un miroir.
– ∼MirrorView ()
Détruit le miroir.

Fonctions membres protégées

– void keyPressEvent (QKeyEvent ∗event)
Permet de faire réagir le miroir à l'action de certaines touches du clavier et de la souris.

Attributs privés

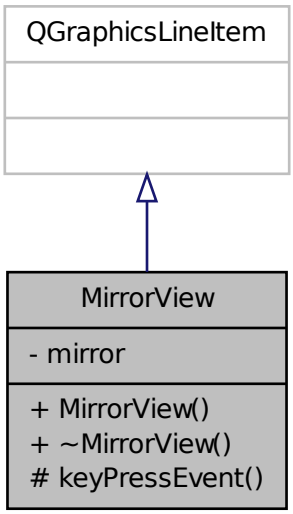
– Mirror ∗ mirror
Le miroir que cette vue représente.

7.12.1 Description détaillée

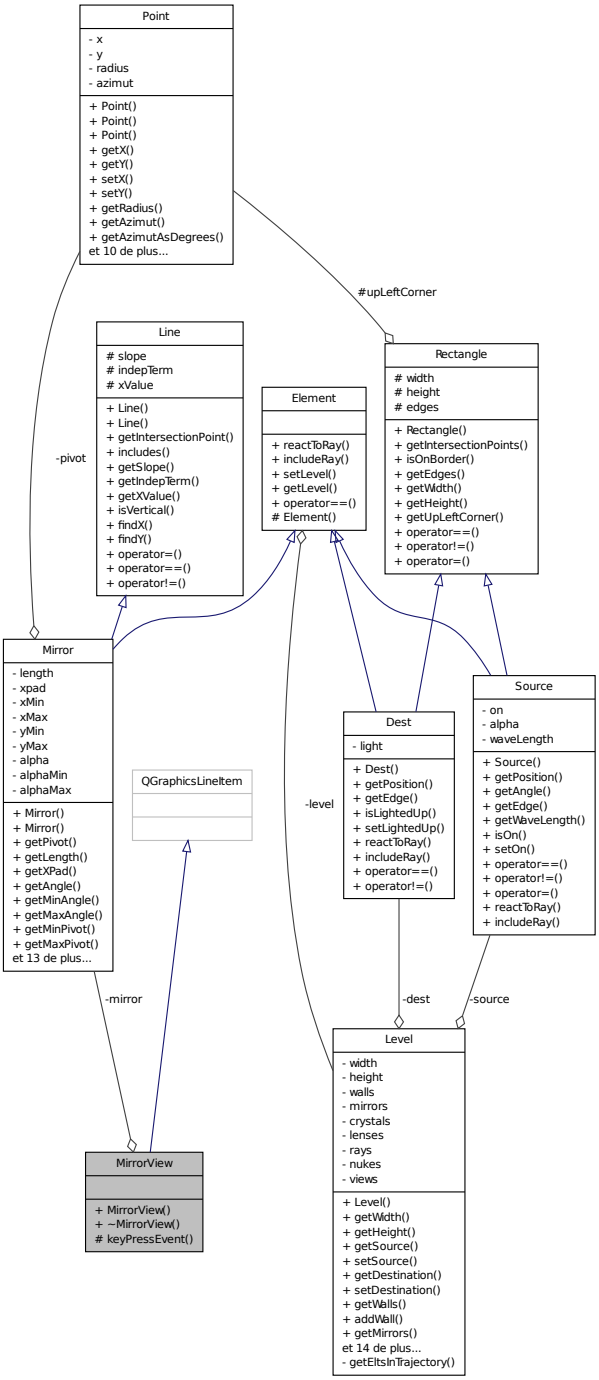
Cette classe représente graphiquement un miroir du jeu permettant d'interagir avec lui à l'aide de la souris et du clavier.

Définition à la ligne 13 du fichier mirrorview.hpp.

Graphe d'héritage de MirrorView :



Graphe de collaboration de MirrorView :



7.12.2 Documentation des constructeurs et destructeur

7.12.2.1 MirrorView : :MirrorView (Mirror ∗ *mirror*)

Construit une vue de miroir lié à un miroir.

Paramètres

mirror	Le miroir lié à cette vue.

7.12.2.2 MirrorView : :~MirrorView ( )

Détruit le miroir.

7.12.3 Documentation des fonctions membres

7.12.3.1 void MirrorView : :keyPressEvent (QKeyEvent * *event*) [protected]

Permet de faire réagir le miroir à l'action de certaines touches du clavier et de la souris.

Paramètres

event	Une évènement "input user".

7.12.4 Documentation des données membres

7.12.4.1 Mirror* MirrorView : :mirror [private]

Le miroir que cette vue représente.

Définition à la ligne 18 du fichier mirrorview.hpp.

La documentation de cette classe a été générée à partir du fichier suivant :
– view/dynamicElements/mirrorview.hpp

7.13 Référence de la classe Nuke

Cette classe modélise les bombes utilisées dans le jeu.

`#include <nuke.hpp>`

Fonctions membres publiques

– Nuke (const Point &, const double)
  Instancie une bombe en une position donnée avec un rayon déterminé.
– const Point & getLocation () const
  Retourne la position de la bombe.
– double getRadius () const
  Retourne le rayon de la bombe.
– bool isLightedUp () const
  Cette méthode permet de savoir si la bombe est illuminée.
– void setLightedUp (const bool)
  Cette méthode permet d'établir un nouvel état d'illumination de la bombe.
– void reactToRay (Ray)
  Réaction à l'exposition d'un rayon.
– Point * includeRay (const Ray &) const
  Renseigne si la bombe est dans la trajectoire du rayon.
– bool operator== (const Nuke &) const
  Permet de savoir si deux bombes sont les mêmes.
– bool operator != (const Nuke &) const
  Permet de savoir si deux bombes sont différentes.

Attributs privés

– bool light
L'état d'illumination d'une bombe.

Membres hérités additionnels

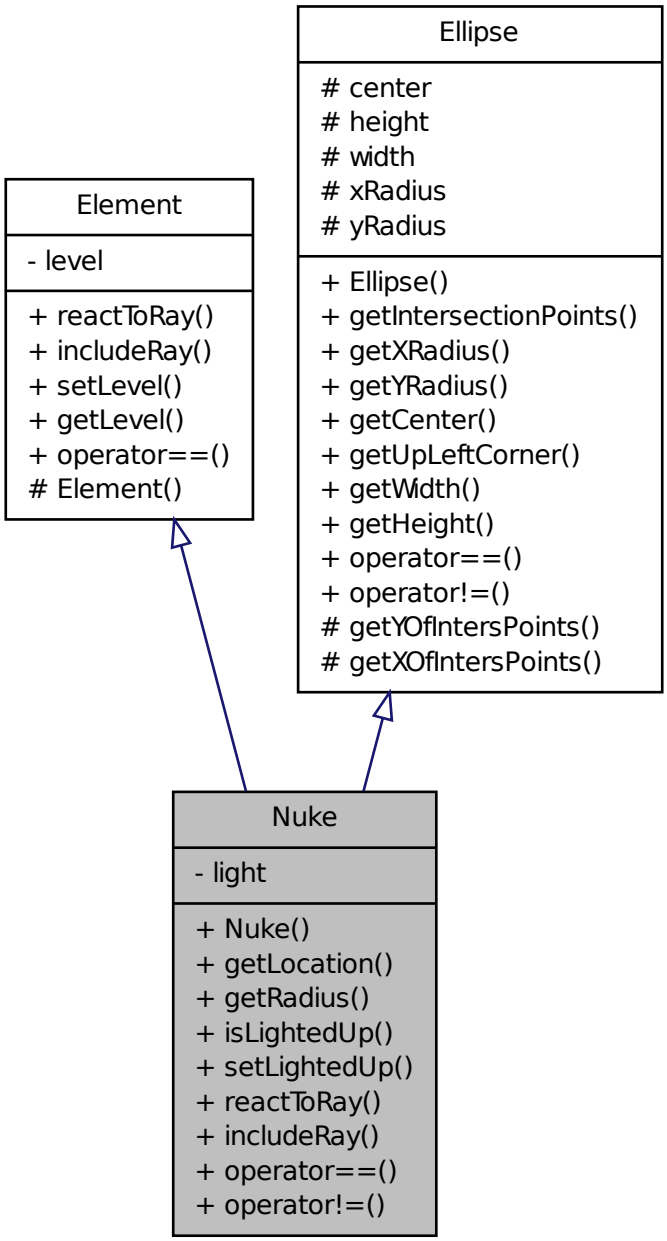
7.13.1 Description détaillée

Cette classe modélise les bombes utilisées dans le jeu.

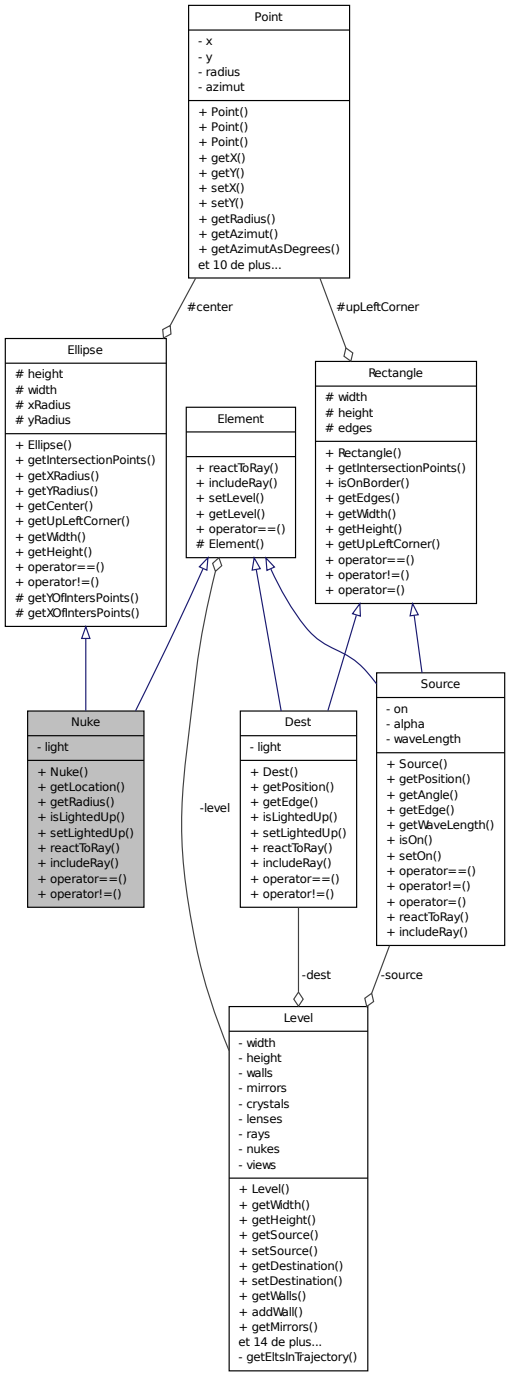
Une bombe est un objet circulaire qui, si illuminé par un rayon, fait perdre la partie au joueur.

Définition à la ligne 17 du fichier nuke.hpp.

Graphe d'héritage de Nuke :



Graphe de collaboration de Nuke :



7.13.2 Documentation des constructeurs et destructeur

7.13.2.1 Nuke : :Nuke (const Point & , const double)

Instancie une bombe en une position donnée avec un rayon déterminé.

Paramètres

position	La position du centre de la bombe.
radius	Le rayon de la bombe.

7.13.3 Documentation des fonctions membres

7.13.3.1 const Point & Nuke : :getLocation ( ) const [inline]

Retourne la position de la bombe.

Renvoie

la position de la bombe.

Définition à la ligne 107 du fichier nuke.hpp.

Références Ellipse : :center.

```
108 {
109     return this->center;
110 }
```

7.13.3.2 double Nuke : :getRadius ( ) const [inline]

Retourne le rayon de la bombe.

Renvoie

le rayon de la bombe.

Définition à la ligne 112 du fichier nuke.hpp.

Références Ellipse : :getHeight().

```
113 {
114     return (this->getHeight() / 2.);
115 }
```

7.13.3.3 Point* Nuke : :includeRay (const Ray &) const [virtual]

Renseigne si la bombe est dans la trajectoire du rayon.

Paramètres

ray	Le rayon.

Renvoie

`true` Si la bombe se trouve dans la trajectoire du rayon entré en paramètre.

Implémente Element.

7.13.3.4 bool Nuke : :isLightedUp ( ) const [inline]

Cette méthode permet de savoir si la bombe est illuminée.

Renvoie

`true` Si la bombe est illuminée, faux sinon.

Définition à la ligne 102 du fichier nuke.hpp.

Références light.

```
103 {
104     return this->light;
105 }
```

7.13.3.5 bool Nuke : :operator != (const Nuke &) const

Permet de savoir si deux bombes sont différentes.

Renvoie

`true` Si les deux bombes sont différentes.

7.13.3.6 bool Nuke : :operator== (const Nuke &) const

Permet de savoir si deux bombes sont les mêmes.

Renvoie

`true` Si les deux bombes sont les mêmes.

7.13.3.7 void Nuke : :reactToRay (Ray) [virtual]

Réaction à l'exposition d'un rayon.

Paramètres

ray	Le rayon.

Implémente Element.

7.13.3.8 void Nuke : :setLightedUp (const bool)

Cette méthode permet d'établir un nouvel état d'illumination de la bombe.

Paramètres

light	Le nouvel état d'illumination de la bombe.

7.13.4 Documentation des données membres

7.13.4.1 bool Nuke : :light [private]

L'état d'illumination d'une bombe.

Définition à la ligne 22 du fichier nuke.hpp.

Référencé par isLightedUp().

La documentation de cette classe a été générée à partir du fichier suivant :
– model/elements/nuke.hpp

7.14 Référence de la classe Point

Cette classe modélise un point de coordonnés dans le plan R^2 sous deux formes :

`#include <point.hpp>`

Fonctions membres publiques

- Point ()=default
  Instancie le point $c(0,0)p(0,0)$.
- Point (const double, const double)
  Instancie le point de coordonnées spécifiées.
- Point (const Point &)
  Instancie un point par copie d'un autre point : constructeur de recopie.
- double getX () const
  Retourne l'abscisse du point.
- double getY () const
  Retourne l'ordonnée du point.
- void setX (const double)
  Déplace le point en l'abscisse donnée.
- void setY (const double)
  Déplace le point en l'ordonnée donnée.
- double getRadius () const
  Permet d'obtenir la distance séparant le point du centre de rotation.
- double getAzimut () const
  Permet d'obtenir l'angle de la coordonnée polaire courante.
- double getAzimutAsDegrees () const
  Permet d'obtenir l'angle de la coordonnée polaire courante exprimée en degrés.
- Point & rotateAround (const Point &, const double)
  Cette méthode change la position du point courant dans le plan par rotation autour du point cartésien passé en paramètre.
- void rotate (const double)
  Effectue une rotation autour de l'origine du plan cartésien.
- void setOrigin (const Point &=Point{0., 0.})
  Considère la position d'un point par rapport à un point quelconque.
- void extend (const double)
  Modifie le point pour lui donner un position de même angle en lui ajoutant un radius.
- void setCartesianLocation (const double, const double)
  Déplace le point en la coordonnée cartésienne donnée.
- void setPolarLocation (const double, const double)
  Déplace le point en la coordonnée polaire donnée.
- double distanceFrom (const Point &) const
  Permet de connaitre la distance séparant le point courant d'un autre.
- Point & operator= (const Point &)
  Permet de copier le contenu d'un point dans un autre point.
- bool operator== (const Point &) const
  Permet de savoir si deux points sont aux même endroit.
- bool operator != (const Point &) const
  Permet de savoir si deux points ne sont pas au même endroit.

Attributs privés

- double x {0.}
  x La distance, sur l'axe des abscisses, du point par rapport à l'origine.
- double y {0.}
  y La distance, sur l'axe des ordonnées, du point par rapport à l'origine.
- double radius {0.}
  radius la distance du point par rapport à l'origine du plan cartésien.
- double azimut {0.}
  azimut le segment de cercle exprimé en radian depuis l'axe horizontal et dans un sens anti-horloger.

7.14.1 Description détaillée

Cette classe modélise un point de coordonnés dans le plan R^2 sous deux formes :

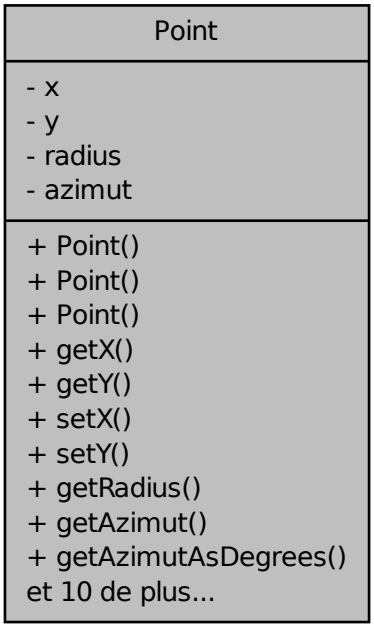
- coordonnées cartésiennes sous la forme $c(x, y)$,

– coordonnées polaires sous la forme $p(r, \alpha)$.

Elle sert à définir la position d'un objet dans l'espace à deux dimensions.

Définition à la ligne 15 du fichier point.hpp.

Graphe de collaboration de Point :



7.14.2 Documentation des constructeurs et destructeur

7.14.2.1 Point : :Point ( ) [default]

Instancie le point $c(0, 0)p(0, 0)$.

7.14.2.2 Point : :Point (const double , const double)

Instancie le point de coordonnées spécifiées.

Paramètres

x	l'abscisse du point
y	l'ordonnée du point

7.14.2.3 Point : :Point (const Point &)

Instancie un point par copie d'un autre point : constructeur de recopie.

7.14.3 Documentation des fonctions membres

7.14.3.1 double Point : :distanceFrom (const Point &) const

Permet de connaitre la distance séparant le point courant d'un autre.

La distance est calculée à l'aide du théorème de Pythagore au triangle rectangle : soient les deux points du plan cartésien reliés formant un segment de droite, en traçant les parallèles aux axes passant par ces points on peut délimiter un triangle rectangle par l'intersection des deux droites. De là, il est simple d'appliquer le théorème de Pythagore aux triangles rectangles : l'hypoténuse au carré = la somme du carré des deux autres cotés $distance(p1, p2)^2 = (p1.x - p2.x)^2 + (p1.y - p2.y)^2$ Dans la bibliothèque standard C++, la fonction std : :hypot de <cmath> nous permet de faire cette opération en lui passant simplement les cotés autre que l'hypoténuse.

Paramètres

point	Un autre point.

Renvoie

La distance séparant deux point.

7.14.3.2 void Point : :extend (const double)

Modifie le point pour lui donner un position de même angle en lui ajoutant un radius.

Paramètres

radius	Un nouveau radius.

7.14.3.3 double Point : :getAzimut ( ) const [inline]

Permet d'obtenir l'angle de la coordonnée polaire courante.

Renvoie

L'amplitude du point polaire courant en radian.

Définition à la ligne 220 du fichier point.hpp.

Références azimut.

```
221 {
222     return this->azimut;
223 }
```

7.14.3.4 double Point : :getAzimutAsDegrees ( ) const

Permet d'obtenir l'angle de la coordonnée polaire courante exprimée en degrés.

Renvoie

L'amplitude du point polaire courant en degré.

7.14.3.5 double Point : :getRadius ( ) const [inline]

Permet d'obtenir la distance séparant le point du centre de rotation.

Renvoie

Le rayon séparant le point polaire de son centre.

Définition à la ligne 215 du fichier point.hpp.

Références radius.

```
216 {
217     return this->radius;
218 }
```

7.14.3.6 double Point : :getX ( ) const [inline]

Retourne l'abscisse du point.

Renvoie

l'abscisse du point.

Définition à la ligne 205 du fichier point.hpp.

Références x.

```
206 {
207     return this->x;
208 }
```

7.14.3.7 double Point : :getY ( ) const [inline]

Retourne l'ordonnée du point.

Renvoie

l'ordonnée du point.

Définition à la ligne 210 du fichier point.hpp.

Références y.

```
211 {
212     return this->y;
213 }
```

7.14.3.8 bool Point : :operator != (const Point &) const

Permet de savoir si deux points ne sont pas au même endroit.

Renvoie

`true` Si les deux points ne sont pas les mêmes.

7.14.3.9 Point& Point : :operator= (const Point &)

Permet de copier le contenu d'un point dans un autre point.

Renvoie

Le point courant modifié.

7.14.3.10 bool Point : :operator== (const Point &) const

Permet de savoir si deux points sont aux même endroit.

Renvoie

`true` Si les deux points ont les même coordonnées.

7.14.3.11 void Point : :rotate (const double)

Effectue une rotation autour de l'origine du plan cartésien.

Paramètres

alpha	L'amplitude de la rotation à effectuer (en radian).

7.14.3.12 Point& Point : :rotateAround (const Point & , const double)

Cette méthode change la position du point courant dans le plan par rotation autour du point cartésien passé en paramètre.

Paramètres

pivot	Le centre autour duquel le point courant doit tourner.
alpha	L'amplitude de la rotation à effectuer (en radian).

Renvoie

Le point courant après rotation.

7.14.3.13 void Point : :setCartesianLocation (const double , const double)

Déplace le point en la coordonnée cartésienne donnée.

Paramètres

x	l'abscisse où déplacer le point.
y	l'ordonnée où déplacer le point.

7.14.3.14 void Point : :setOrigin (const Point & = Point{0., 0.})

Considère la position d'un point par rapport à un point quelconque.

Paramètres

origin	La nouvelle origine du plan. Si ce paramètre est omis, l'origine du plan est rétabli.

7.14.3.15 void Point : :setPolarLocation (const double , const double)

Déplace le point en la coordonnée polaire donnée.

Paramètres

radius	La distance séparant le point de l'origine.
azimut	L'angle, en radian, selon le cercle trigonométrique.

7.14.3.16 void Point : :setX (const double)

Déplace le point en l'abscisse donnée.

Paramètres

x	l'abscisse où déplacer le point.

7.14.3.17 void Point : :setY (const double)

Déplace le point en l'ordonnée donnée.

Paramètres

y	l'ordonnée où déplacer le point.

7.14.4 Documentation des données membres

7.14.4.1 double Point : :azimut {0.} [private]

azimut le segment de cercle exprimé en radian depuis l'axe horizontal et dans un sens anti-horloger.

Définition à la ligne 41 du fichier point.hpp.

Référencé par getAzimut().

7.14.4.2 double Point : :radius {0.} [private]

radius la distance du point par rapport à l'origine du plan cartésien.

Définition à la ligne 35 du fichier point.hpp.

Référencé par getRadius().

7.14.4.3 double Point : :x {0.} [private]

x La distance, sur l'axe des abscisses, du point par rapport à l'origine.

Définition à la ligne 23 du fichier point.hpp.

Référencé par getX().

7.14.4.4 double Point : :y {0.} [private]

y La distance, sur l'axe des ordonnées, du point par rapport à l'origine.

Définition à la ligne 29 du fichier point.hpp.

Référencé par getY().

La documentation de cette classe a été générée à partir du fichier suivant :
– model/geometry/point.hpp

7.15 Référence de la classe Ray

Cette classe modélise les rayons lumineux, concept central du jeu.

`#include <ray.hpp>`

Fonctions membres publiques

- Ray (const Point, double, int=Ray : :WL_DFT)
  Créer un nouveau rayon.
- const Point & getStart () const
  Retourne le début du rayon.
- const Point & getEnd () const
  Retourne la fin du rayon.
- int getWaveLength () const
  Retourne la longueur d'onde du rayon.
- double getAlpha () const
  Permet de connaitre l'angle du rayon.
- void setStart (const Point &)
  Change la coordonnée du début du rayon.
- void setEnd (const Point &)
  hange la coordonnée de la fin du rayon.
- void setWaveLength (const int)
  hange la longueur d'onde du rayon.
- bool isInTrajectory (const Point &) const
  Cette méthode permet de savoir si le point passé en paramètre est bien dans la trajectoire du rayon courant à l'aide de la représentation polaire des points.
- bool operator== (const Ray &) const
  Permet de savoir si deux rayons sont les mêmes.
- bool operator != (const Ray &) const
  Permet de savoir si deux rayons sont différents.

Attributs publics statiques

- static const int WL_MIN {360}
  Longueur d'onde minimum autorisée pour un rayon lumineux.
- static const int WL_MAX {830}
  Longueur d'onde maximum autorisée pour un rayon lumineux.
- static const int WL_DFT {600}
  Longueur d'onde par défaut pour un rayon lumineux.

Attributs protégés

- Point start
  Le point de départ du segment de droite représentant le rayon.
- Point end
  Le point d'arrivé du segment de droite représentant le rayon.
- double alpha
  L'angle de tir du rayon selon le cercle trigonométrique usuel.
- int waveLength
  La longueur d'onde du rayon.

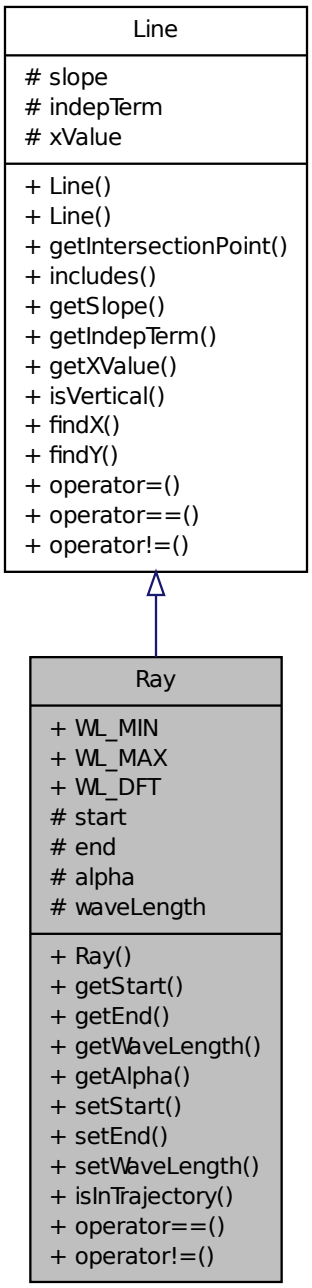
7.15.1 Description détaillée

Cette classe modélise les rayons lumineux, concept central du jeu.

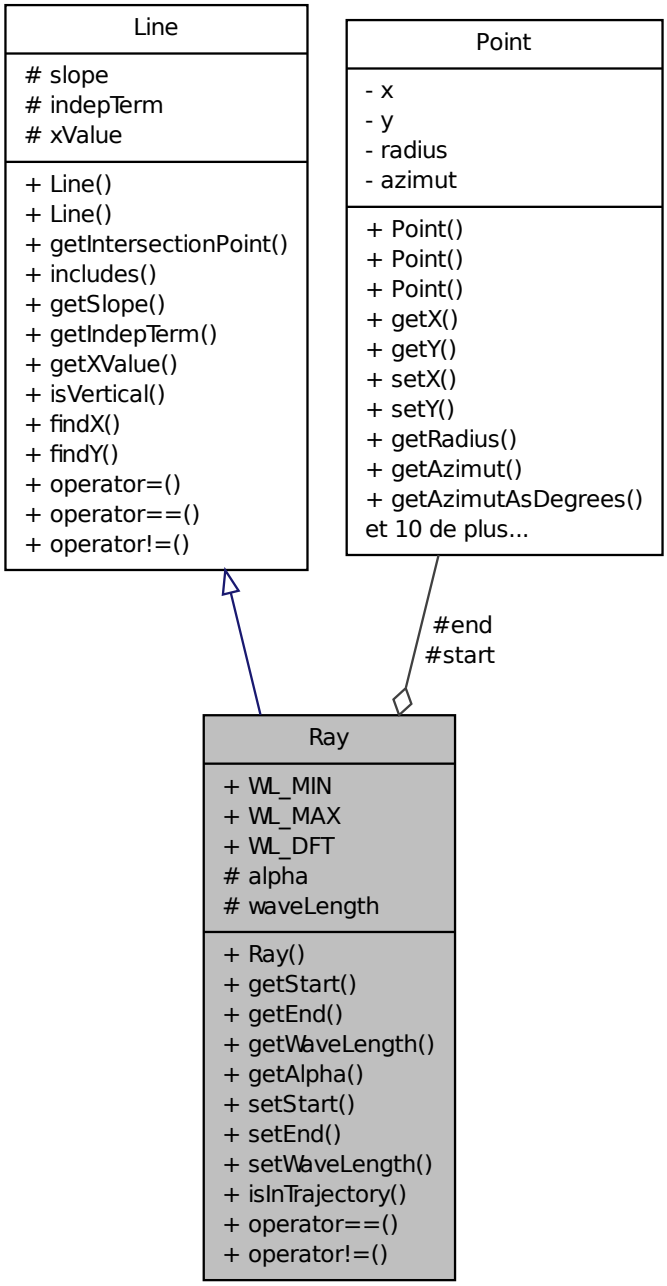
Un rayon lumineux est un segment de droite muni d'une longueur d'onde.

Définition à la ligne 15 du fichier ray.hpp.

Graphe d'héritage de Ray :



Graphe de collaboration de Ray :



7.15.2 Documentation des constructeurs et destructeur

7.15.2.1 Ray : :Ray (const Point , double , int = Ray::WL_DFT)

Créer un nouveau rayon.

7.15.3 Documentation des fonctions membres

7.15.3.1 double Ray : :getAlpha ( ) const [inline]

Permet de connaitre l'angle du rayon.

Renvoie

L'angle du rayon courant.

Définition à la ligne 169 du fichier ray.hpp.

Références alpha.

```
170 {
171     return this->alpha;
172 }
```

7.15.3.2 const Point & Ray : :getEnd ( ) const [inline]

Retourne la fin du rayon.

Renvoie

la fin du rayon.

Définition à la ligne 159 du fichier ray.hpp.

Références end.

```
160 {
161     return this->end;
162 }
```

7.15.3.3 const Point & Ray : :getStart ( ) const [inline]

Retourne le début du rayon.

Renvoie

le début du rayon.

Définition à la ligne 154 du fichier ray.hpp.

Références start.

```
155 {
156     return this->start;
157 }
```

7.15.3.4 int Ray : :getWaveLength ( ) const [inline]

Retourne la longueur d'onde du rayon.

Renvoie

la longueur d'onde du rayon.

Définition à la ligne 164 du fichier ray.hpp.

Références waveLength.

```
165 {
166     return this->waveLength;
167 }
```

7.15.3.5 bool Ray : :isInTrajectory (const Point &) const

Cette méthode permet de savoir si le point passé en paramètre est bien dans la trajectoire du rayon courant à l'aide de la représentation polaire des points.

Renvoie

`true` si le point passé en paramètre est dans la trajectoire.

7.15.3.6 bool Ray : :operator != (const Ray &) const

Permet de savoir si deux rayons sont différents.

Renvoie

`true` Si deux rayons sont différents.

7.15.3.7 bool Ray : :operator== (const Ray &) const

Permet de savoir si deux rayons sont les mêmes.

Renvoie

`true` Si deux rayons sont les même.

7.15.3.8 void Ray : :setEnd (const Point &)

hange la coordonnée de la fin du rayon.

Paramètres

end	La nouvelle coordonnée de la fin du rayon.

7.15.3.9 void Ray : :setStart (const Point &)

Change la coordonnée du début du rayon.

Paramètres

start	La nouvelle coordonnée du début du rayon.

7.15.3.10 void Ray : :setWaveLength (const int)

hange la longueur d'onde du rayon.

Si la longueur d'onde spécifiée est en dehors des limites autorisées, la longueur d'onde vaudra la borne la plus proche. La longueur d'onde doit être comprise entre 360 et 830 nm.

Paramètres

waveLength	La nouvelle longueur d'onde du rayon

7.15.4 Documentation des données membres

7.15.4.1 double Ray : :alpha [protected]

L'angle de tir du rayon selon le cercle trigonométrique usuel.

Définition à la ligne 33 du fichier ray.hpp.

Référencé par getAlpha().

7.15.4.2 Point Ray : :end [protected]

Le point d'arrivé du segment de droite représentant le rayon.

Définition à la ligne 28 du fichier ray.hpp.

Référencé par getEnd().

7.15.4.3 Point Ray : :start [protected]

Le point de départ du segment de droite représentant le rayon.

Définition à la ligne 23 du fichier ray.hpp.

Référencé par getStart().

7.15.4.4 int Ray : :waveLength [protected]

La longueur d'onde du rayon.

Définition à la ligne 38 du fichier ray.hpp.

Référencé par getWaveLength().

7.15.4.5 const int Ray : :WL_DFT {600} [static]

Longueur d'onde par défaut pour un rayon lumineux.

Cette valeur correspond à la longueur d'onde (en nm) de la couleur orangé-rouge du spectre visible de la lumière.

Définition à la ligne 61 du fichier ray.hpp.

7.15.4.6 const int Ray : :WL_MAX {830} [static]

Longueur d'onde maximum autorisée pour un rayon lumineux.

Cette valeur correspond à la longueur d'onde maximum (en nm) du spectre visible de la lumière.

Définition à la ligne 54 du fichier ray.hpp.

7.15.4.7 const int Ray : :WL_MIN {360} [static]

Longueur d'onde minimum autorisée pour un rayon lumineux.

Cette valeur correspond à la longueur d'onde minimum (en nm) du spectre visible de la lumière.

Définition à la ligne 47 du fichier ray.hpp.

La documentation de cette classe a été générée à partir du fichier suivant :
– model/elements/ray.hpp

7.16 Référence de la classe Rectangle

Le rectangle est objet géométrique, du plan, à quatre coté parallèles deux à deux.

`#include <rectangle.hpp>`

Fonctions membres publiques

- Rectangle (double, double, const Point &)
  Permet de construire un nouveau rectangle initialisé.
- std : :vector< Point > getIntersectionPoints (const Line &) const
  Permet d'obtenir les points d'intersection entre le rectangle et la droite entrée en paramètre.
- bool isOnBorder (const Point &) const
  Renseigne si un point se trouve sur la bordure du rectangle.
- std : :vector< Line > getEdges () const
  Permet d'obtenir les côtés du rectangle sous forme de droites.
- virtual double getWidth () const
  Permet d'obtenir la longueur du rectangle.
- virtual double getHeight () const
  Permet d'obtenir la hauteur du rectangle.
- Point getUpLeftCorner () const
  Permet d'obtenir les coordonnées du coté supérieur du rectangle.
- bool operator== (const Rectangle &) const
  Permet de savoir si deux rectangles sont identiques.
- bool operator != (const Rectangle &) const
  Permet de savoir si deux rectangles sont différents.
- Rectangle & operator= (const Rectangle &)
  Permet de copier un rectangle dans un autre.

Attributs protégés

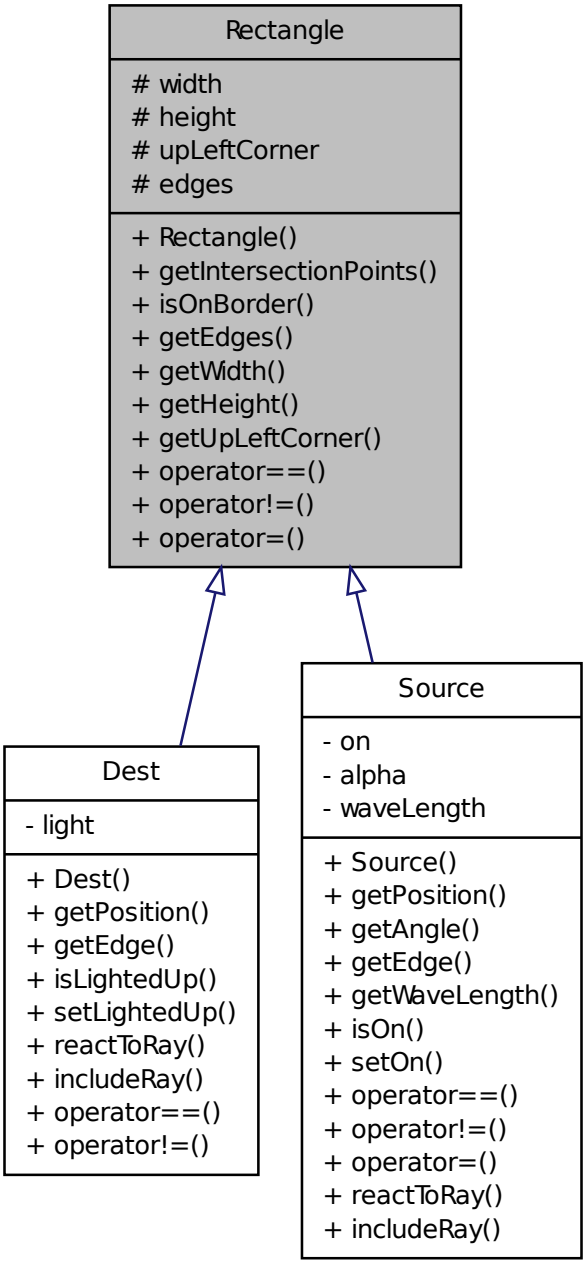
- double width
  La largeur du rectangle.
- double height
  La longueur du rectangle.
- Point upLeftCorner
  La position du coin supérieur gauche du rectangle.
- std : :vector< Line > edges
  Les équations des 4 droites composant le rectangle.

7.16.1 Description détaillée

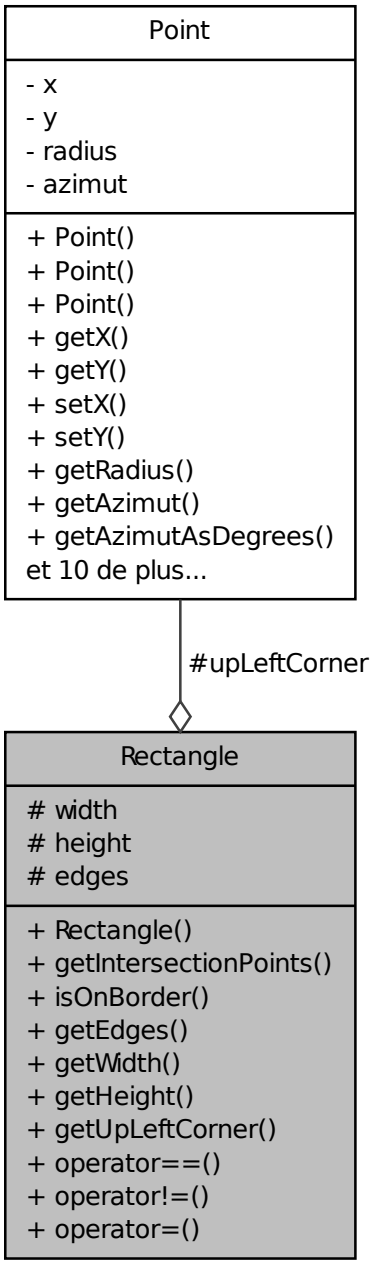
Le rectangle est objet géométrique, du plan, à quatre coté parallèles deux à deux.

Définition à la ligne 14 du fichier rectangle.hpp.

Graphe d'héritage de Rectangle :



Graphe de collaboration de Rectangle :



7.16.2 Documentation des constructeurs et destructeur

7.16.2.1 Rectangle : :Rectangle (double , double , const Point &)

Permet de construire un nouveau rectangle initialisé.

Paramètres

width	Largeur du rectangle.
height	Hauteur du rectangle.
upLeftCorner	Côté supérieur gauche du rectangle.

7.16.3 Documentation des fonctions membres

7.16.3.1 std : :vector< Line > Rectangle : :getEdges ( ) const [inline]

Permet d'obtenir les côtés du rectangle sous forme de droites.

Renvoie

Les côtés du rectangle sous forme de droites.

Définition à la ligne 132 du fichier rectangle.hpp.

Références edges.

```
133 {
134     return this->edges;
135 }
```

7.16.3.2 double Rectangle : :getHeight ( ) const [inline], [virtual]

Permet d'obtenir la hauteur du rectangle.

Renvoie

La hauteur du rectangle.

Définition à la ligne 142 du fichier rectangle.hpp.

Références height.

```
143 {
144     return this->height;
145 }
```

7.16.3.3 std : :vector<Point> Rectangle : :getIntersectionPoints (const Line &) const

Permet d'obtenir les points d'intersection entre le rectangle et la droite entrée en paramètre.

Paramètres

line	Droite dont on désire obtenir le point d'intersection avec la droite courante.

Renvoie

Un vecteur contenant les points d'intersection entre la droite entrée en paramètre et le rectangle.

7.16.3.4 Point Rectangle : :getUpLeftCorner ( ) const [inline]

Permet d'obtenir les coordonnées du coté supérieur du rectangle.

Renvoie

Les coordonnées du coté supérieur du rectangle.

Définition à la ligne 147 du fichier rectangle.hpp.

Références upLeftCorner.

```
148 {
149     return this->upLeftCorner;
150 }
```

7.16.3.5 double Rectangle : :getWidth ( ) const [inline],[virtual]

Permet d'obtenir la longueur du rectangle.

Renvoie

La longueur du rectangle.

Définition à la ligne 137 du fichier rectangle.hpp.

Références width.

```
138 {
139     return this->width;
140 }
```

7.16.3.6 bool Rectangle : :isOnBorder (const Point &) const

Renseigne si un point se trouve sur la bordure du rectangle.

Paramètres

point	Point dont on désire savoir s'il est inclus sur la bordure du rectangle.

Renvoie

`true` Si le Point entré en paramètre est inclus sur la bordure du rectangle.

7.16.3.7 bool Rectangle : :operator != (const Rectangle &) const

Permet de savoir si deux rectangles sont différents.

Renvoie

`true` Si les deux rectangles sont différents.

7.16.3.8 Rectangle& Rectangle : :operator= (const Rectangle &)

Permet de copier un rectangle dans un autre.

Renvoie

Le rectangle courant modifié.

7.16.3.9 bool Rectangle : :operator== (const Rectangle &) const

Permet de savoir si deux rectangles sont identiques.

Renvoie

`true` Si les deux rectangles sont identiques.

7.16.4 Documentation des données membres

7.16.4.1 std : :vector<Line> Rectangle : :edges [protected]

Les équations des 4 droites composant le rectangle.

Définition à la ligne 37 du fichier rectangle.hpp.

Référencé par getEdges().

7.16.4.2 double Rectangle : :height [protected]

La longueur du rectangle.

Définition à la ligne 27 du fichier rectangle.hpp.

Référencé par Dest : :getEdge(), et getHeight().

7.16.4.3 Point Rectangle : :upLeftCorner [protected]

La position du coin supérieur gauche du rectangle.

Définition à la ligne 32 du fichier rectangle.hpp.

Référencé par Dest : :getPosition(), Source : :getPosition(), et getUpLeftCorner().

7.16.4.4 double Rectangle : :width [protected]

La largeur du rectangle.

Définition à la ligne 22 du fichier rectangle.hpp.

Référencé par Source : :getEdge(), et getWidth().

La documentation de cette classe a été générée à partir du fichier suivant :
– model/geometry/rectangle.hpp

7.17 Référence de la classe Source

Modélise la source lumineuse utilisée dans le jeu.

`#include <source.hpp>`

Fonctions membres publiques

– Source (const Point &, const int, const double, const int)
  Instancie une nouvelle source de position, côté et longueur d'onde donnée.
– const Point & getPosition () const
  Retourne la coordonnée du coin supérieur gauche du carré modélisant la destination.
– double getAngle () const
  Retourne l'angle du rayon émis.
– int getEdge () const

Retourne la longueur du côté du carré.
- int getWaveLength () const
  Retourne la longueur d'onde du rayon émis.
- bool isOn () const
  Permet de savoir l'état d'émission de la source.
- void setOn (const bool)
  Allume ou éteint la source.
- bool operator== (const Source &) const
  Permet de savoir si deux sources sont les mêmes.
- bool operator != (const Source &) const
  Permet de savoir si deux sources sont différentes.
- Source & operator= (const Source &)
  Permet de copier la source en paramètre dans la source locale.
- void reactToRay (Ray)
  Cette méthode est la réaction de la source face à un rayon.
- Point * includeRay (const Ray &) const
  Cette méthode permet de savoir si la source comprend un point.

Attributs privés

- bool on
  État d'émission de la source.
- double alpha
  L'angle, en radian, d'émission de la source lumineuse.
- int waveLength
  La longueur d'onde du rayon tiré par la source.

Membres hérités additionnels

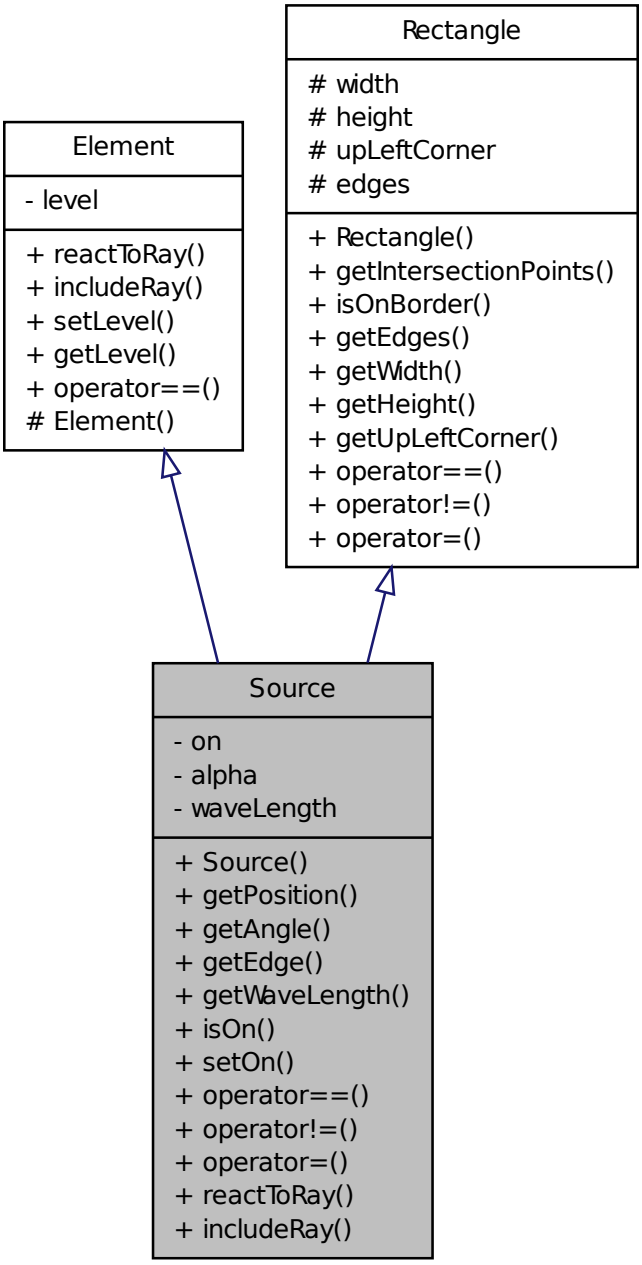
7.17.1 Description détaillée

Modélise la source lumineuse utilisée dans le jeu.

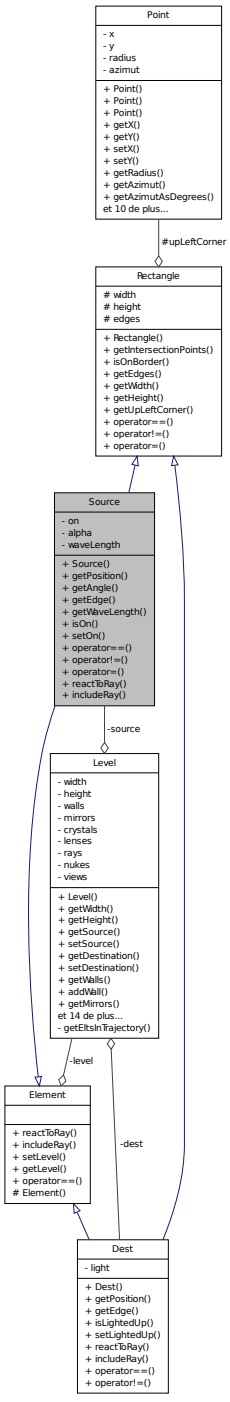
La source est un objet carré qui, si allumée, émet un rayon lumineux de longueur d'onde donnée dont l'angle ne peut pas être changé. Le rayon lumineux est émis depuis la position, i.e., le coin supérieur gauche, de la source.

Définition à la ligne 19 du fichier source.hpp.

Graphe d'héritage de Source :



Graphe de collaboration de Source :



7.17.2 Documentation des constructeurs et destructeur

7.17.2.1 Source : :Source (const Point & , const int , const double , const int)

Instancie une nouvelle source de position, côté et longueur d'onde donnée.

La position dénote la coordonnée du coin supérieur gauche du carré modélisant la source. La source est initialement éteinte. Si la longueur d'onde du rayon lumineux émis n'est pas comprise entre 360 nm et 830 nm, elle est réglée

sur 600 nm.

Paramètres

position	La position du coin supérieur gauche de la source.
edge	La longueur du côté du carré modélisant la source.
waveLength	La longueur d'onde du rayon lumineux émis.

Voir également

Ray : :WL_MIN
Ray : :WL_MAX
Ray : :WL_DFT

7.17.3 Documentation des fonctions membres

7.17.3.1 double Source : :getAngle ( ) const [inline]

Retourne l'angle du rayon émis.

Renvoie

l'angle du rayon émis.

Définition à la ligne 146 du fichier source.hpp.

Références alpha.

```
147 {
148     return this->alpha;
149 }
```

7.17.3.2 int Source : :getEdge ( ) const [inline]

Retourne la longueur du côté du carré.

Renvoie

la longueur du côté du carré.

Définition à la ligne 151 du fichier source.hpp.

Références Rectangle : :width.

```
152 {
153     return this->width;
154 }
```

7.17.3.3 const Point & Source : :getPosition ( ) const [inline]

Retourne la coordonnée du coin supérieur gauche du carré modélisant la destination.

Renvoie

La coordonnée du coin supérieur gauche du carré modélisant la source.

Définition à la ligne 141 du fichier source.hpp.

Références Rectangle : :upLeftCorner.

```
142 {
143     return this->upLeftCorner;
144 }
```

7.17.3.4 int Source : :getWaveLength ( ) const [inline]

Retourne la longueur d'onde du rayon émis.

Renvoie

la longueur d'onde du rayon émis.

Définition à la ligne 156 du fichier source.hpp.

Références waveLength.

```
157 {
158     return this->waveLength;
159 }
```

7.17.3.5 Point* Source : :includeRay (const Ray &) const [virtual]

Cette méthode permet de savoir si la source comprend un point.

Renvoie

`nullptr` dans tout les cas, la source est un objet qui ne réagit pas.

Implémente Element.

7.17.3.6 bool Source : :isOn ( ) const [inline]

Permet de savoir l'état d'émission de la source.

Renvoie

`true` Si la source émet un rayon lumineux.

Définition à la ligne 161 du fichier source.hpp.

Références on.

```
162 {
163     return this->on;
164 }
```

7.17.3.7 bool Source : :operator != (const Source &) const

Permet de savoir si deux sources sont différentes.

Renvoie

`true` Si deux sources sont différentes.

7.17.3.8 Source& Source : :operator= (const Source &)

Permet de copier la source en paramètre dans la source locale.

Renvoie

La source locale modifiée.

7.17.3.9 bool Source : :operator== (const Source &) const

Permet de savoir si deux sources sont les mêmes.

Renvoie

`true` Si deux sources sont les mêmes.

7.17.3.10 void Source : :reactToRay (Ray) [virtual]

Cette méthode est la réaction de la source face à un rayon.

Celui- ci ne fait rien.

Implémente Element.

7.17.3.11 void Source : :setOn (const bool)

Allume ou éteint la source.

Paramètres

on	Le nouvel état de la source.

7.17.4 Documentation des données membres

7.17.4.1 double Source : :alpha [private]

L'angle, en radian, d'émission de la source lumineuse.

Définition à la ligne 29 du fichier source.hpp.

Référencé par getAngle().

7.17.4.2 bool Source : :on [private]

État d'émission de la source.

Définition à la ligne 24 du fichier source.hpp.

Référencé par isOn().

7.17.4.3 int Source : :waveLength [private]

La longueur d'onde du rayon tiré par la source.

Définition à la ligne 34 du fichier source.hpp.

Référencé par getWaveLength().

La documentation de cette classe a été générée à partir du fichier suivant :
– model/elements/source.hpp

7.18 Référence de la classe SourceView

Cette classe permet de représenter graphiquement une source, lui permettant de communiquer les actions utilisateurs.

`#include <sourceview.hpp>`

Fonctions membres publiques

- SourceView (Source *source)
  Permet de créer une vue liée à une source.
- ~SourceView ()
- void switchSource ()
  Permet de changer l'état de la source.

Fonctions membres protégées

- void mousePressEvent (QGraphicsSceneMouseEvent *event)
  Permet de réagir sur le modèle lors d'un "input user".

Attributs privés

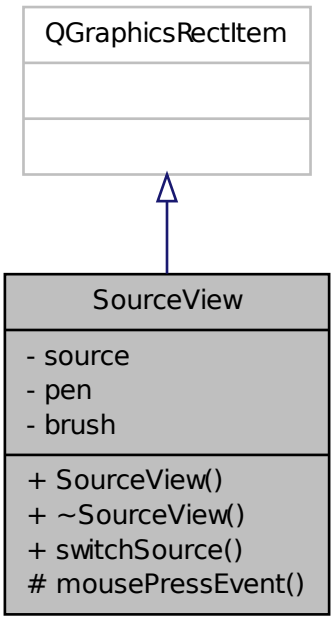
- Source * source
  La source représentée par cette vue.
- QPen pen
  Les paramètres visuels du trait de la source.
- QBrush brush
  Les paramètres visuels du plein de la source.

7.18.1 Description détaillée

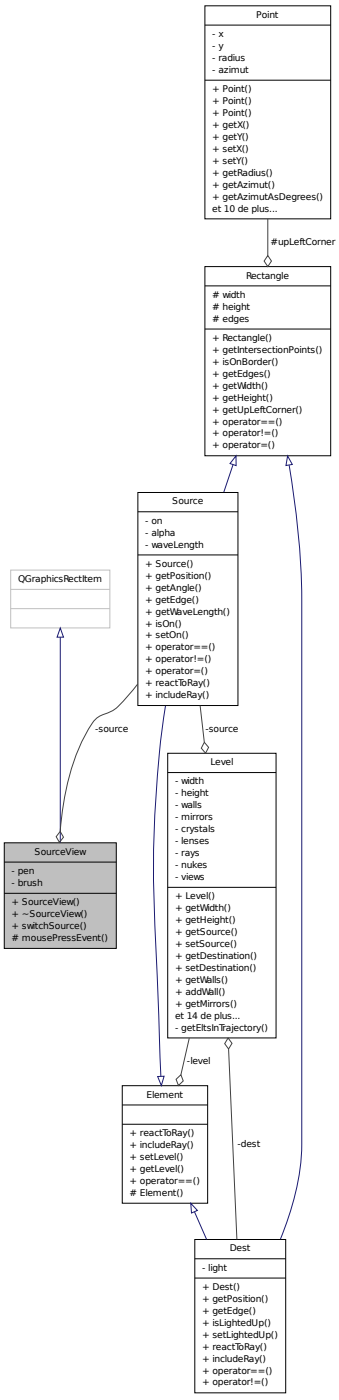
Cette classe permet de représenter graphiquement une source, lui permettant de communiquer les actions utilisateurs.

Définition à la ligne 14 du fichier sourceview.hpp.

Graphe d'héritage de SourceView :



Graphe de collaboration de SourceView :



7.18.2 Documentation des constructeurs et destructeur

7.18.2.1 SourceView : :SourceView (Source ∗ *source*)

Permet de créer une vue liée à une source.

Paramètres

source	La source liée à cette vue.

7.18.2.2 SourceView : :~SourceView ( )

7.18.3 Documentation des fonctions membres

7.18.3.1 void SourceView : :mousePressEvent (QGraphicsSceneMouseEvent * *event*) [protected]

Permet de réagir sur le modèle lors d'un "input user".

Paramètres

event	Un évènement "input user".

7.18.3.2 void SourceView : :switchSource ( )

Permet de changer l'état de la source.

7.18.4 Documentation des données membres

7.18.4.1 QBrush SourceView : :brush [private]

Les paramètres visuels du plein de la source.

Définition à la ligne 29 du fichier sourceview.hpp.

7.18.4.2 QPen SourceView : :pen [private]

Les paramètres visuels du trait de la source.

Définition à la ligne 24 du fichier sourceview.hpp.

7.18.4.3 Source* SourceView : :source [private]

La source représentée par cette vue.

Définition à la ligne 19 du fichier sourceview.hpp.

La documentation de cette classe a été générée à partir du fichier suivant :
– view/dynamicElements/sourceview.hpp

7.19 Référence de la classe StarlightException

Cette classe représente une exception spécifique au jeu Starlight.

`#include <starlightexception.hpp>`

Fonctions membres publiques

– StarlightException (std : :string)
Construit une nouvelle erreur inhérente au jeu.
– std : :string getMessage () const
Permet d'obtenir le message d'erreur de l'exception.

– const char ∗ what () const throw ()
Permet d'afficher l'erreur en cas d'erreur.

Attributs privés

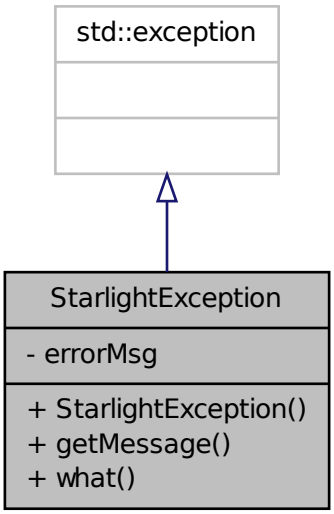
– std : :string errorMsg
Le message d'erreur de l'exception lancée.

7.19.1 Description détaillée

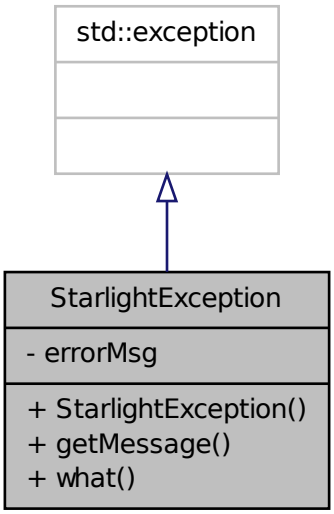
Cette classe représente une exception spécifique au jeu Starlight.

Définition à la ligne 10 du fichier starlightexception.hpp.

Graphe d'héritage de StarlightException :



Graphe de collaboration de StarlightException :



7.19.2 Documentation des constructeurs et destructeur

7.19.2.1 StarlightException : :StarlightException (std : :string)

Construit une nouvelle erreur inhérente au jeu.

Paramètres

errorMsg	Message expliquant l'erreur.

7.19.3 Documentation des fonctions membres

7.19.3.1 std : :string StarlightException : :getMessage ( ) const [inline]

Permet d'obtenir le message d'erreur de l'exception.

Renvoie

Le message d'erreur de l'exception.

Définition à la ligne 45 du fichier starlightexception.hpp.

Références errorMsg.

```
46 {
47     return this->errorMsg;
48 }
```

7.19.3.2 const char * StarlightException : :what ( ) const throw) [inline]

Permet d'afficher l'erreur en cas d'erreur.

Renvoie

Le message d'erreur.

Définition à la ligne 50 du fichier starlightexception.hpp.

```
51 {
52     return this->errorMsg.c_str();
53 }
```

7.19.4 Documentation des données membres

7.19.4.1 std : :string StarlightException : :errorMsg [private]

Le message d'erreur de l'exception lancée.

Définition à la ligne 18 du fichier starlightexception.hpp.

Référencé par getMessage().

La documentation de cette classe a été générée à partir du fichier suivant :
– model/exception/starlightexception.hpp

7.20 Référence de la classe Wall

Cette classe modélise les murs utilisés dans le jeu.

`#include <wall.hpp>`

Fonctions membres publiques

– Wall (const Point &, const Point &)
Instancie un mur.
– const Point & getStart () const
Retourne le début du mur.
– const Point & getEnd () const
Retourne la fin du mur.
– void reactToRay (Ray)
Réaction à l'exposition d'un rayon.
– Point ∗ includeRay (const Ray &) const
Renseigne si le mur est dans la trajectoire du rayon.
– bool operator== (const Wall &) const
Permet de savoir si deux murs sont identiques.
– bool operator != (const Wall &) const
Permet de savoir si deux murs sont différents.

Attributs privés

– Point start
Le point de départ du segment de droite représentant le mur.
– Point end
Le point d'arrivé du segment de droite représentant le mur.

Membres hérités additionnels

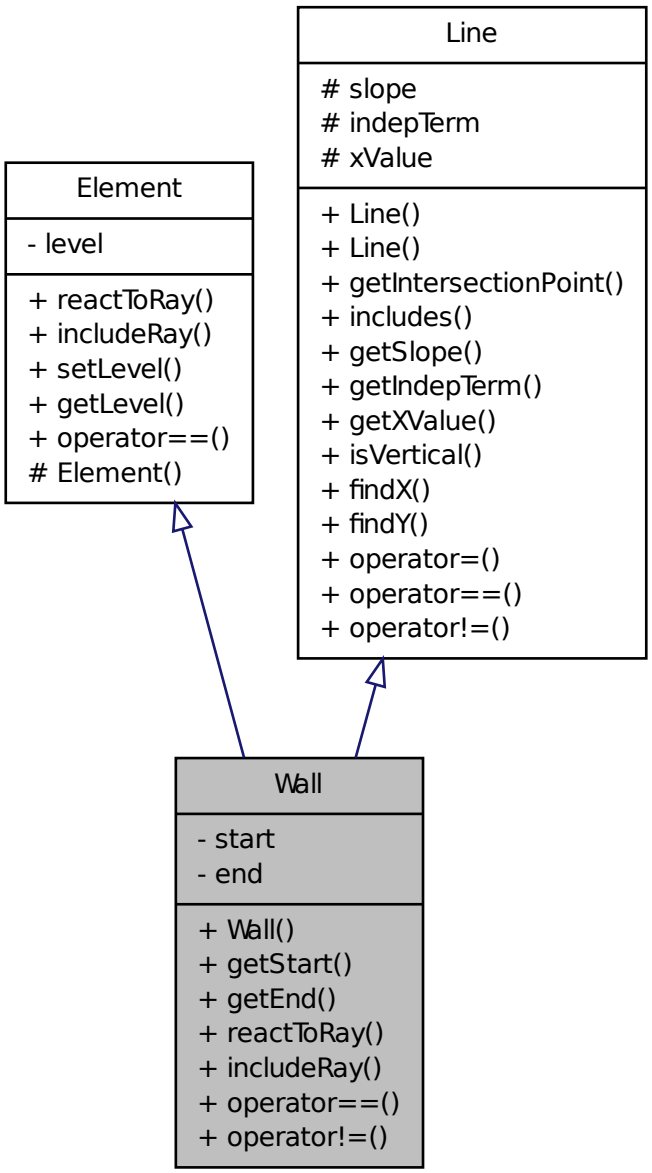
7.20.1 Description détaillée

Cette classe modélise les murs utilisés dans le jeu.

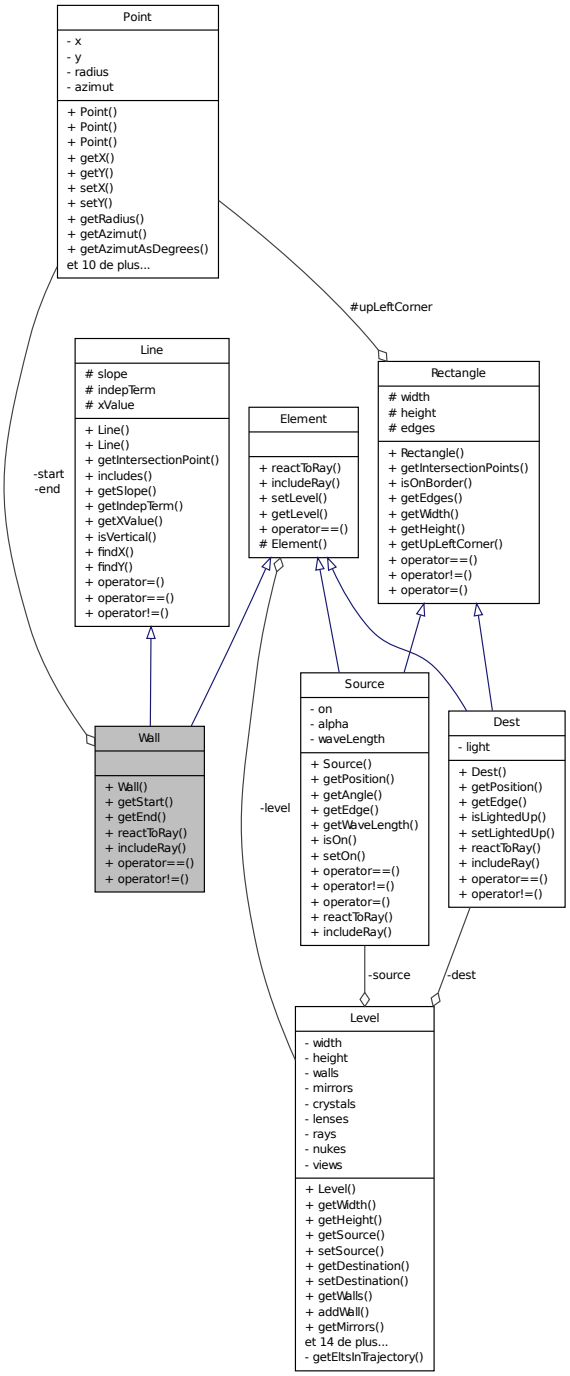
Les murs sont des segments de droite qui ne réfléchissent pas la lumière.

Définition à la ligne 16 du fichier wall.hpp.

Graphe d'héritage de Wall :



Graphe de collaboration de Wall :



7.20.2 Documentation des constructeurs et destructeur

7.20.2.1 Wall : :Wall (const Point & , const Point &)

Instancie un mur.

Paramètres

start	Le début du mur.
end	La fin du mur.

7.20.3 Documentation des fonctions membres

7.20.3.1 const Point & Wall : :getEnd ( ) const [inline]

Retourne la fin du mur.

Renvoie

La fin du mur.

Définition à la ligne 97 du fichier wall.hpp.

Références end.

```
98 {
99     return this->end;
100 }
```

7.20.3.2 const Point & Wall : :getStart ( ) const [inline]

Retourne le début du mur.

Renvoie

Le début du mur.

Définition à la ligne 92 du fichier wall.hpp.

Références start.

```
93 {
94     return this->start;
95 }
```

7.20.3.3 Point* Wall : :includeRay (const Ray &) const [virtual]

Renseigne si le mur est dans la trajectoire du rayon.

Paramètres

ray	Le rayon.

Renvoie

`true` Si la mur se trouve dans la trajectoire du rayon entré en paramètre.

Implémente Element.

7.20.3.4 bool Wall : :operator != (const Wall &) const

Permet de savoir si deux murs sont différents.

Renvoie

`true` Si les murs sont différents.

7.20.3.5 bool Wall : :operator== (const Wall &) const

Permet de savoir si deux murs sont identiques.

Renvoie

`true` Si les murs sont les même.

7.20.3.6 void Wall : :reactToRay (Ray) [virtual]

Réaction à l'exposition d'un rayon.

Paramètres

ray	Le rayon.

Implémente Element.

7.20.4 Documentation des données membres

7.20.4.1 Point Wall : :end [private]

Le point d'arrivé du segment de droite représentant le mur.

Définition à la ligne 26 du fichier wall.hpp.

Référencé par getEnd().

7.20.4.2 Point Wall : :start [private]

Le point de départ du segment de droite représentant le mur.

Définition à la ligne 21 du fichier wall.hpp.

Référencé par getStart().

La documentation de cette classe a été générée à partir du fichier suivant :
– model/elements/wall.hpp

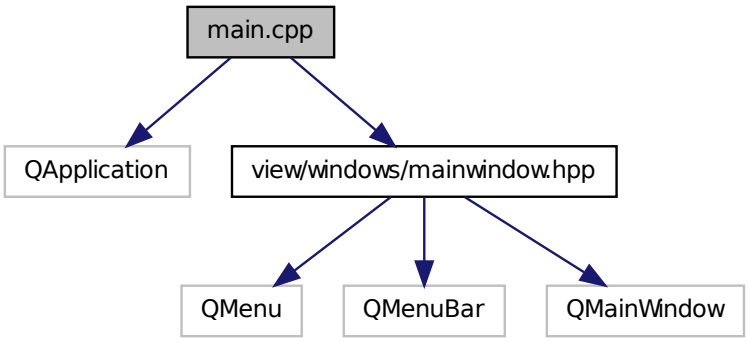
Chapitre 8

Documentation des fichiers

8.1 Référence du fichier main.cpp

```
#include <QApplication>
#include "view/windows/mainwindow.hpp"
```

Graphe des dépendances par inclusion de main.cpp :



Fonctions

– int main (int argc, char ∗argv[ ])

8.1.1 Documentation des fonctions

8.1.1.1 int main (int *argc,* char ∗ *argv[ ]*)

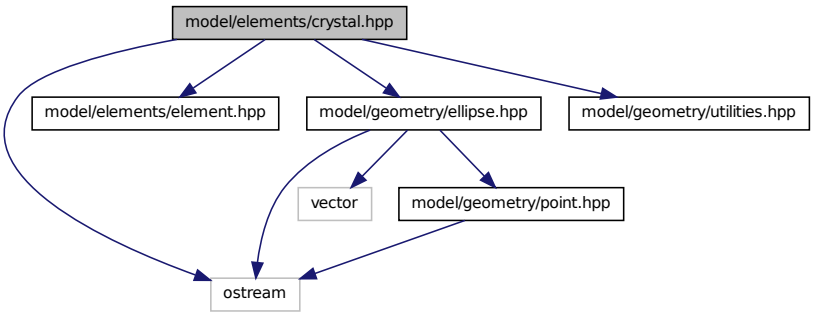
Définition à la ligne 5 du fichier main.cpp.

```
6 {
7     QApplication a(argc, argv);
8     MainWindow w;
9
10     w.show();
11
12     return a.exec();
13 }
```

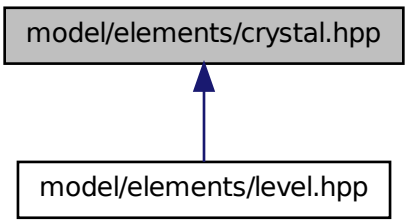
8.2 Référence du fichier model/elements/crystal.hpp

```
#include <ostream>
#include "model/elements/element.hpp"
#include "model/geometry/ellipse.hpp"
#include "model/geometry/utilities.hpp"
```

Graphe des dépendances par inclusion de crystal.hpp :



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Classes

– class Crystal
Cette classe amplifie les cristaux utilisés dans le jeu.

Fonctions

– std : :ostream & operator<< (std : :ostream &, const Crystal &)
Définition, externe, de l'opérateur permettant de produire un affichage formaté.

8.2.1 Documentation des fonctions

8.2.1.1 std : :ostream& operator<< (std : :ostream & , const Crystal &)

Définition, externe, de l'opérateur permettant de produire un affichage formaté.

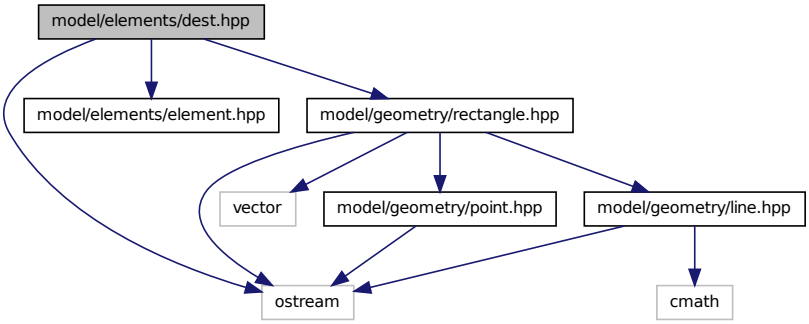
Renvoie

Le ostream rempli de la chaine formatée représentant le Crystal en paramètre.

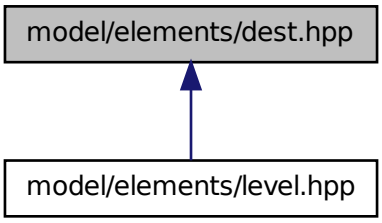
8.3 Référence du fichier model/elements/dest.hpp

```
#include <ostream>
#include "model/elements/element.hpp"
#include "model/geometry/rectangle.hpp"
```

Graphe des dépendances par inclusion de dest.hpp :



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Classes

– class Dest
Cette classe modélise la destination utilisée dans le jeu.

Fonctions

– std : :ostream & operator<< (std : :ostream &, const Dest &)
Définition, externe, de l'opérateur permettant de produire un affichage formaté.

8.3.1 Documentation des fonctions

8.3.1.1 std : :ostream& operator<< (std : :ostream & , const Dest &)

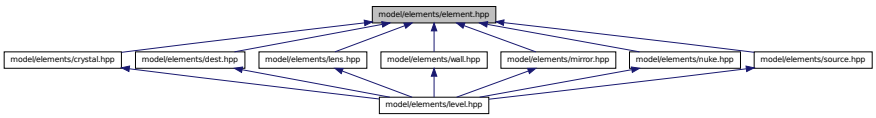
Définition, externe, de l'opérateur permettant de produire un affichage formaté.

Renvoie

Le ostream rempli de la chaine formatée représentant la Dest en paramètre.

8.4 Référence du fichier model/elements/element.hpp

Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



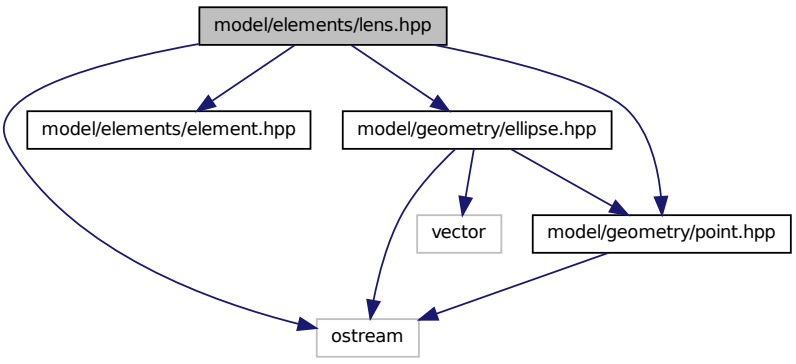
Classes

– class Element
Un élément est un composant du jeu se devant de communiquer son état au niveau le gérant.

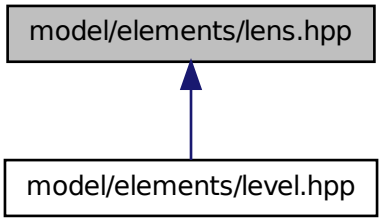
8.5 Référence du fichier model/elements/lens.hpp

```
#include <ostream>
#include "model/elements/element.hpp"
#include "model/geometry/ellipse.hpp"
#include "model/geometry/point.hpp"
```

Graphe des dépendances par inclusion de lens.hpp :



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Classes

– class Lens
Cette classe modélise les lentilles utilisées dans le jeu.

Fonctions

– std : :ostream & operator<< (std : :ostream &, const Lens &)
Définition, externe, de l'opérateur permettant de produire un affichage formaté.

8.5.1 Documentation des fonctions

8.5.1.1 std : :ostream& operator<< (std : :ostream & , const Lens &)

Définition, externe, de l'opérateur permettant de produire un affichage formaté.

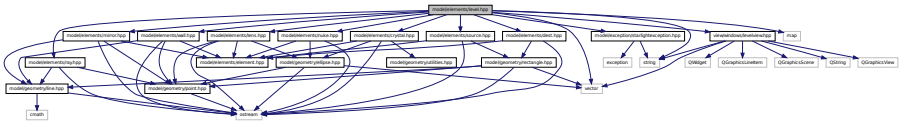
Renvoie

Le ostream rempli de la chaine formatée représentant la Lens en paramètre.

8.6 Référence du fichier model/elements/level.hpp

```
#include <vector>
#include <map>
#include <string>
#include "model/exception/starlightexception.hpp"
#include "model/elements/wall.hpp"
#include "model/elements/mirror.hpp"
#include "model/elements/crystal.hpp"
#include "model/elements/lens.hpp"
#include "model/elements/nuke.hpp"
#include "model/elements/source.hpp"
#include "model/elements/dest.hpp"
#include "model/elements/ray.hpp"
#include "view/windows/levelview.hpp"
```

Graphe des dépendances par inclusion de level.hpp :



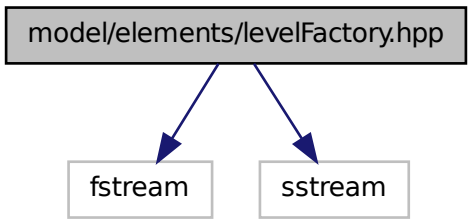
Classes

– class Level
Modélise une carte telle qu'utilisée dans le jeu.

8.7 Référence du fichier model/elements/levelFactory.hpp

```
#include <fstream>
#include <sstream>
```

Graphe des dépendances par inclusion de levelFactory.hpp :



Espaces de nommage

– levelFactory
Fonctions utilitaires permettant divers éléments du jeu à partir d'un fichier .lvl.

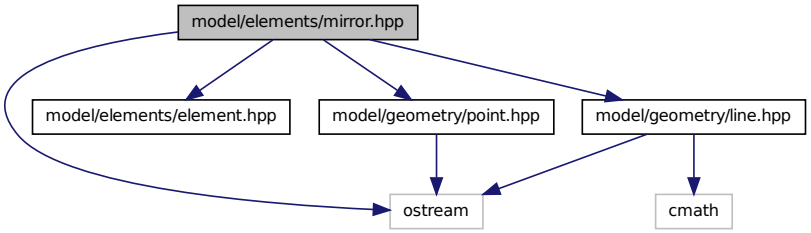
Fonctions

– Level ∗ levelFactory : :getLevelFromFile (std : :string)
Permet d'obtenir une référence vers une nouvelle carte initialisée à partir d'un fichier .level.
– Source levelFactory : :getSource (std : :ifstream &)
Permet d' obtenir une source à partir d'un fichier .lvl déjà ouvert.
– Dest levelFactory : :getDestination (std : :ifstream &)
Permet d' obtenir une destination à partir d'un fichier .lvl déjà ouvert.
– Crystal levelFactory : :getCrystal (std : :ifstream &)
Permet d' obtenir un crystal à partir d'un fichier .lvl déjà ouvert.
– Lens levelFactory : :getLens (std : :ifstream &)
Permet d' obtenir une lentille à partir d'un fichier .lvl déjà ouvert.
– Wall levelFactory : :getWall (std : :ifstream &)
Permet d' obtenir un mur à partir d'un fichier .lvl déjà ouvert.
– Nuke levelFactory : :getNuke (std : :ifstream &)
Permet d' obtenir une bombe à partir d'un fichier .lvl déjà ouvert.
– Mirror levelFactory : :getMirror (std : :ifstream &)
Permet d' obtenir un mirroir à partir d'un fichier .lvl déjà ouvert.

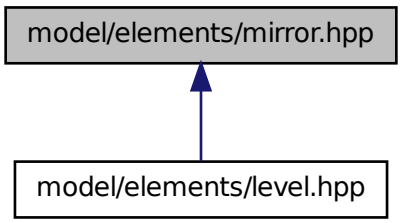
8.8 Référence du fichier model/elements/mirror.hpp

```
#include <ostream>
#include "model/elements/element.hpp"
#include "model/geometry/line.hpp"
#include "model/geometry/point.hpp"
```

Graphe des dépendances par inclusion de mirror.hpp :



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Classes

– class Mirror
Cette classe modélise les miroirs utilisés dans le jeu.

Fonctions

– std : :ostream & operator<< (std : :ostream &, const Mirror &)
Définition, externe, de l'opérateur permettant de produire un affichage formaté.

8.8.1 Documentation des fonctions

8.8.1.1 std : :ostream& operator<< (std : :ostream & , const Mirror &)

Définition, externe, de l'opérateur permettant de produire un affichage formaté.

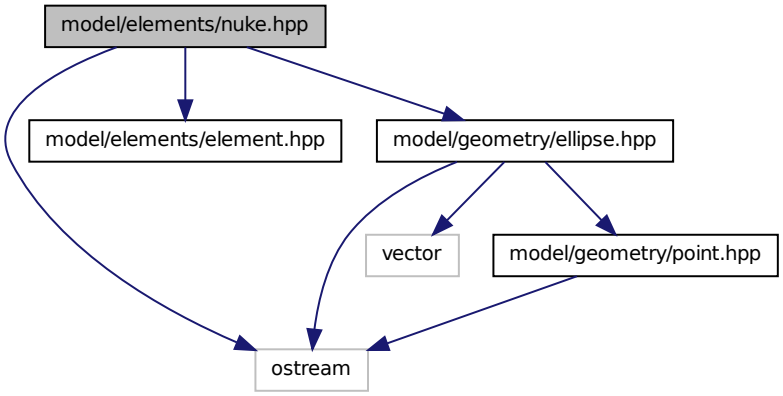
Renvoie

Le ostream rempli de la chaine formatée représentant le Mirror en paramètre.

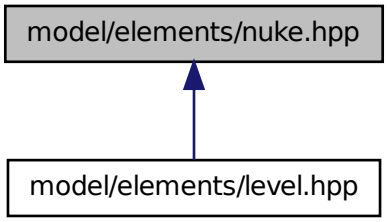
8.9 Référence du fichier model/elements/nuke.hpp

```
#include <ostream>
#include "model/elements/element.hpp"
#include "model/geometry/ellipse.hpp"
```

Graphe des dépendances par inclusion de nuke.hpp :



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Classes

– class Nuke
Cette classe modélise les bombes utilisées dans le jeu.

Fonctions

– std : :ostream & operator<< (std : :ostream &, const Nuke &)
Définition, externe, de l'opérateur permettant de produire un affichage formaté.

8.9.1 Documentation des fonctions

8.9.1.1 std : :ostream& operator<< (std : :ostream & , const Nuke &)

Définition, externe, de l'opérateur permettant de produire un affichage formaté.

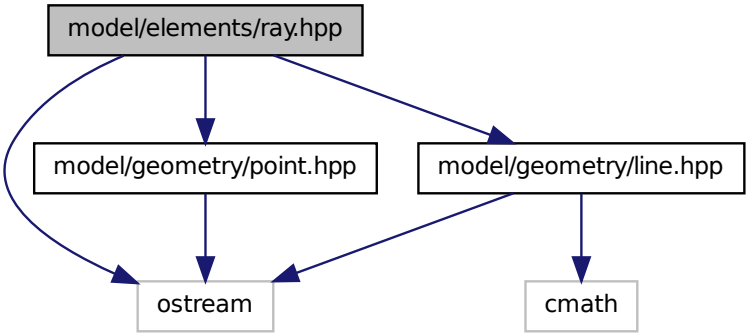
Renvoie

Le ostream rempli de la chaine formatée représentant le Nuke en paramètre.

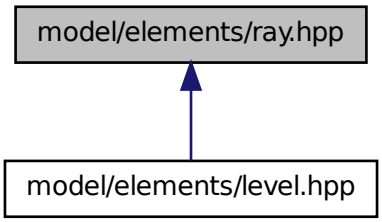
8.10 Référence du fichier model/elements/ray.hpp

```
#include <ostream>
#include "model/geometry/point.hpp"
#include "model/geometry/line.hpp"
```

Graphe des dépendances par inclusion de ray.hpp :



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Classes

– class Ray
Cette classe modélise les rayons lumineux, concept central du jeu.

Fonctions

– std : :ostream & operator<< (std : :ostream &, const Ray &)
Définition, externe, de l'opérateur permettant de produire un affichage formaté.

8.10.1 Documentation des fonctions

8.10.1.1 std : :ostream& operator<< (std : :ostream & , const Ray &)

Définition, externe, de l'opérateur permettant de produire un affichage formaté.

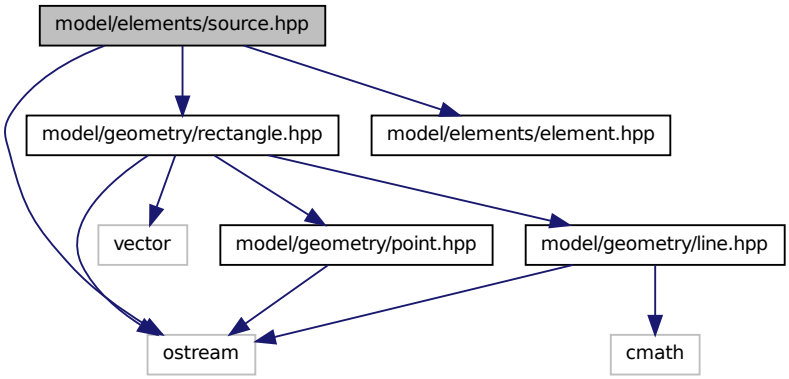
Renvoie

Le ostream rempli de la chaine formatée représentant le Ray en paramètre.

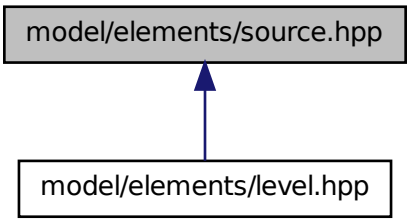
8.11 Référence du fichier model/elements/source.hpp

```
#include <ostream>
#include "model/geometry/rectangle.hpp"
#include "model/elements/element.hpp"
```

Graphe des dépendances par inclusion de source.hpp :



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Classes

– class Source
Modélise la source lumineuse utilisée dans le jeu.

Fonctions

– std : :ostream & operator<< (std : :ostream &, const Source &)
Définition, externe, de l'opérateur permettant de produire un affichage formaté.

8.11.1 Documentation des fonctions

8.11.1.1 std : :ostream& operator<< (std : :ostream & , const Source &)

Définition, externe, de l'opérateur permettant de produire un affichage formaté.

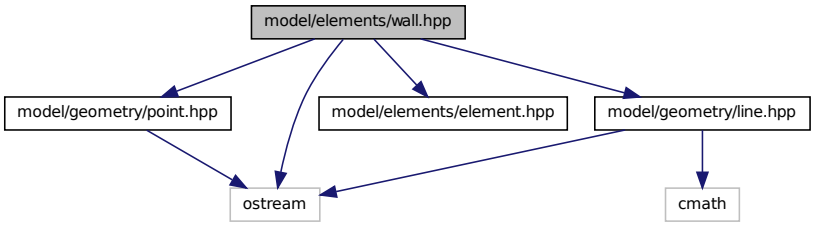
Renvoie

Le ostream rempli de la chaine formatée représentant la Source en paramètre.

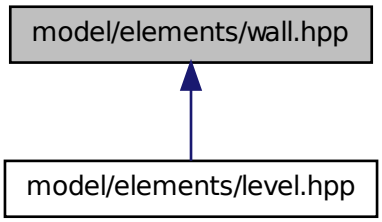
8.12 Référence du fichier model/elements/wall.hpp

```
#include <ostream>
#include "model/geometry/line.hpp"
#include "model/elements/element.hpp"
#include "model/geometry/point.hpp"
```

Graphe des dépendances par inclusion de wall.hpp :



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Classes

– class Wall
Cette classe modélise les murs utilisés dans le jeu.

Fonctions

– std : :ostream & operator<< (std : :ostream &, const Wall &)
Définition, externe, de l'opérateur permettant de produire un affichage formaté.

8.12.1 Documentation des fonctions

8.12.1.1 std : :ostream& operator<< (std : :ostream & , const Wall &)

Définition, externe, de l'opérateur permettant de produire un affichage formaté.

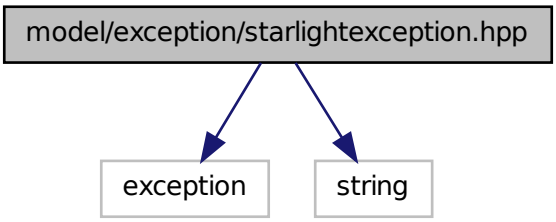
Renvoie

Le ostream rempli de la chaine formatée représentant le Wall en paramètre.

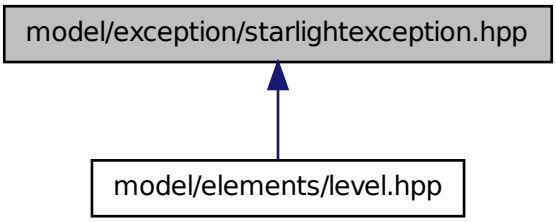
8.13 Référence du fichier model/exception/starlightexception.hpp

```
#include <exception>
#include <string>
```

Graphe des dépendances par inclusion de starlightexception.hpp :



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



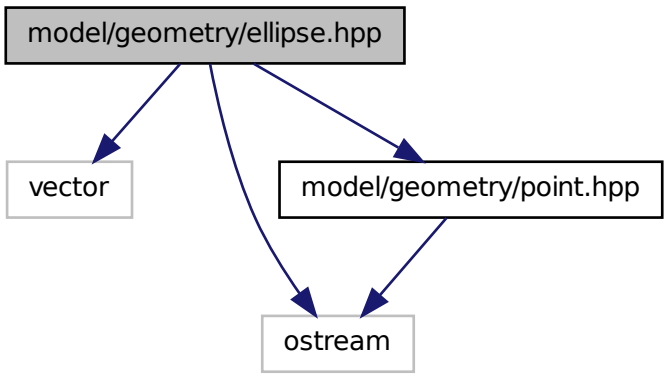
Classes

– class StarlightException
Cette classe représente une exception spécifique au jeu Starlight.

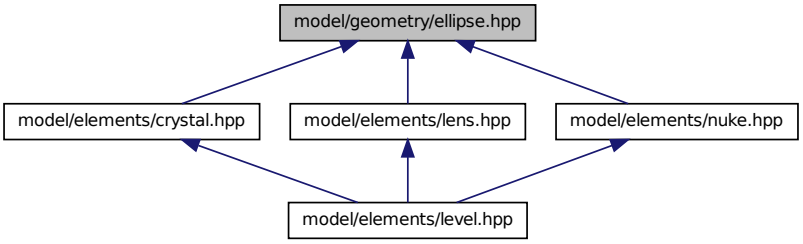
8.14 Référence du fichier model/geometry/ellipse.hpp

```
#include <vector>
#include <ostream>
#include "model/geometry/point.hpp"
```

Graphe des dépendances par inclusion de ellipse.hpp :



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Classes

– class Ellipse
Représente un cercle sous la forme ; $circle \equiv x^2/xRadius + y^2/yRadius = 1.$

Fonctions

– std : :ostream & operator<< (std : :ostream &, const Ellipse &)
Définition, externe, de l'opérateur permettant de produire un affichage formaté.

8.14.1 Documentation des fonctions

8.14.1.1 std : :ostream& operator<< (std : :ostream & , const Ellipse &)

Définition, externe, de l'opérateur permettant de produire un affichage formaté.

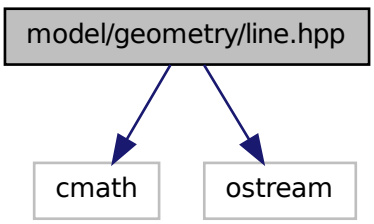
Renvoie

Le ostream rempli de la chaine formatée représentant l'ellipse en paramètre.

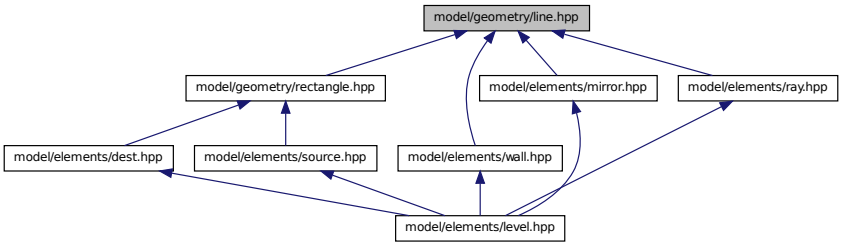
8.15 Référence du fichier model/geometry/line.hpp

```
#include <cmath>
#include <ostream>
```

Graphe des dépendances par inclusion de line.hpp :



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Classes

– class Line
Représente une droite sous la forme de son équation complète ; $eq \equiv y = slope \cdot x + indepTerm.$

Fonctions

– std : :ostream & operator<< (std : :ostream &, const Line &)
Définition, externe, de l'opérateur permettant de produire un affichage formaté.

8.15.1 Documentation des fonctions

8.15.1.1 std : :ostream& operator<< (std : :ostream & , const Line &)

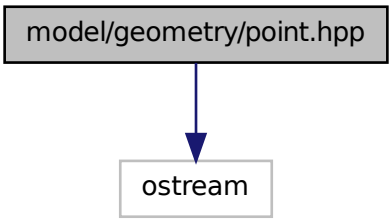
Définition, externe, de l'opérateur permettant de produire un affichage formaté.

Renvoie

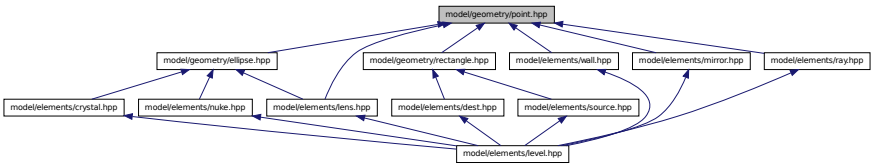
Le ostream rempli de la chaine formatée représentant la Ligne en paramètre.

8.16 Référence du fichier model/geometry/point.hpp

```
#include <ostream>
```
Graphe des dépendances par inclusion de point.hpp :



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Classes

– class Point
Cette classe modélise un point de coordonnés dans le plan R^2 sous deux formes :

Fonctions

– std : :ostream & operator<< (std : :ostream &, const Point &)
Définition, externe, de l'opérateur permettant de produire un affichage formaté.

8.16.1 Documentation des fonctions

8.16.1.1 std : :ostream& operator<< (std : :ostream & , const Point &)

Définition, externe, de l'opérateur permettant de produire un affichage formaté.

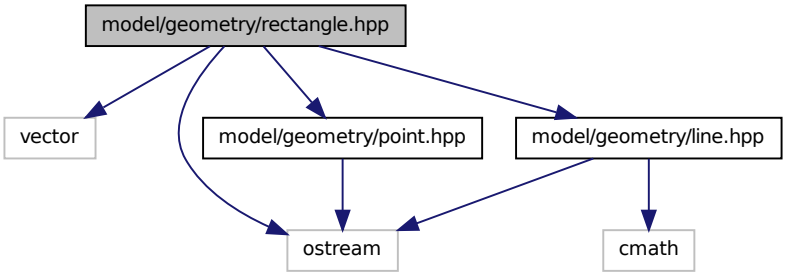
Renvoie

Le ostream rempli de la chaine formatée représentant le Point en paramètre.

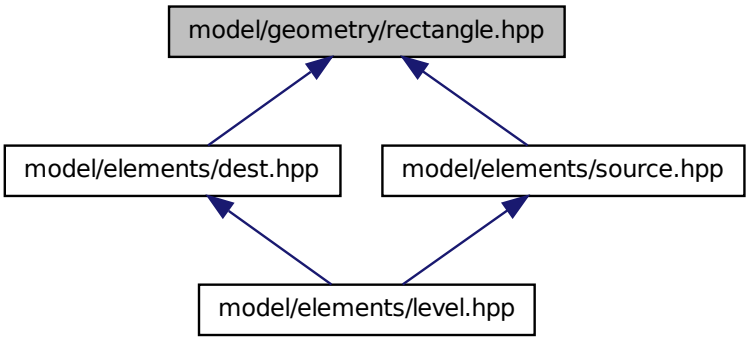
8.17 Référence du fichier model/geometry/rectangle.hpp

```
#include <vector>
#include <ostream>
#include "model/geometry/point.hpp"
#include "model/geometry/line.hpp"
```

Graphe des dépendances par inclusion de rectangle.hpp :



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Classes

– class Rectangle
Le rectangle est objet géométrique, du plan, à quatre coté parallèles deux à deux.

Fonctions

– std : :ostream & operator<< (std : :ostream &, const Rectangle &)
Définition, externe, de l'opérateur permettant de produire un affichage formaté.

8.17.1 Documentation des fonctions

8.17.1.1 std : :ostream& operator<< (std : :ostream & , const Rectangle &)

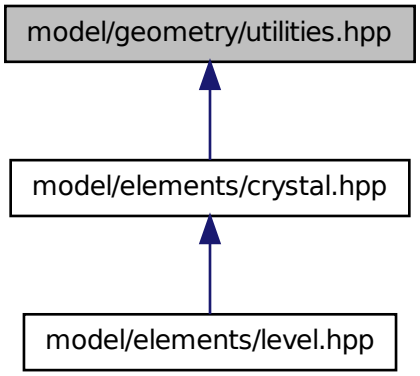
Définition, externe, de l'opérateur permettant de produire un affichage formaté.

Renvoie

Le ostream rempli de la chaine formatée représentant le Rectangle en paramètre.

8.18 Référence du fichier model/geometry/utilities.hpp

Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Espaces de nommage

– utilities
Diverse fonctions utilitaires de géométrie.

Fonctions

– bool utilities : :secondDegreeEquationSolver (double, double, double, double $*$, double $*$)
Permet de trouver les racines (si elles existe) d'une fonction du deuxième degré de forme $ax + bx + c$.
– double utilities : :radianAsDegree (const double)
Permet de trouver l'angle en degré d'un angle en radian.
– double utilities : :radianAsDegree0to360 (const double)
Permet de trouver l'angle en degré, entre 0 et 360, d'un angle en radian.
– bool utilities : :equals (const double, const double, const double=utilities : :EPSILON)
Cette méthode permet de savoir si deux double sont égaux avec une marge d'erreur Epsilon passée en paramètre ou imposée par défaut à $\epsilon = 10^{-7}$.
– int utilities : :round (const double)
Cette méthode cast un double en int on l'ayant au préalable arrondi à l'unité la plus proche (0.5).
– bool utilities : :greaterOrEquals (const double, const double, const double=utilities : :EPSILON)
Cette méthode permet de vérifier l'inégalité $nb_1 \geq nb_2$ sur deux nombres réels avec une marge d'erreur Epsilon passée en paramètre ou imposée par défaut à $\epsilon = 10^{-7}$.
– bool utilities : :lessOrEquals (const double, const double, const double=utilities : :EPSILON)
Cette méthode permet de vérifier l'inégalité $nb_1 \leq nb_2$ sur deux nombres réels avec une marge d'erreur Epsilon passée en paramètre ou imposée par défaut à $\epsilon = 10^{-7}$.
– double utilities : :degreeToRadian (const double)
Cette méthode permet de transformer des degrés en radian.
– double utilities : :slopeFromPoints (const Point &, const Point &)
Permet de trouver la pente d'une droite formée par deux points.
– bool utilities : :isHalfPiPlusNPi (const double)
Permet de savoir si l'angle, en radian, vaut $\frac{\pi}{2} + n \cdot (2 \cdot \pi)$.
– double utilities : :tan (const double)
Permet d'avoir la valeur trigonométrique tangente d'un angle ou l'infini si $angle = \frac{\pi}{2} + n \cdot 2 \cdot \pi$.

– double utilities : :absoluteAngle (const double)
  Permet d'avoir l'angle "absolu" de celui passé en paramètre, [0, PI_2].
– double utilities : :inZeroTwoPi (const double)
  Permet de cadrer un angle dans un intervalle [0 ; 2PI[.

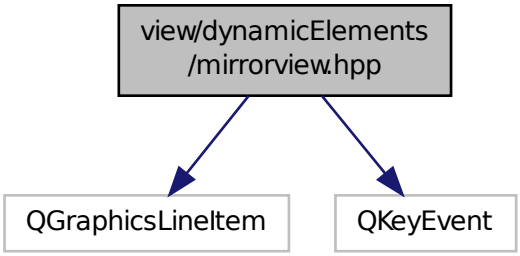
Variables

– const double utilities : :PI {3.14159265358979323846}
  PI Représentation de la constante PI sur 26 décimales.
– const double utilities : :PI_2 {1.57079632679489661923}
  PI_2 Représentation de la constante PI/2 sur 26 décimales.
– const double utilities : :PI_4 {0.78539816339744830961 6}
  PI_4 Représentation de la constante PI/4 sur 26 décimales.
– const double utilities : :EPSILON {10E-7}
  EPSILON Représentation de la marge d'erreur maximale acceptée.
– const double utilities : :INF {1./0.}
  INF Représente une division impossible.

8.19 Référence du fichier view/dynamicElements/mirrorview.hpp

```
#include <QGraphicsLineItem>
#include <QKeyEvent>
```

Graphe des dépendances par inclusion de mirrorview.hpp :



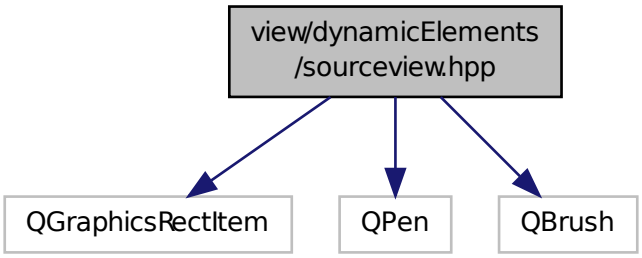
Classes

– class MirrorView
  Cette classe représente graphiquement un miroir du jeu permettant d'interagir avec lui à l'aide de la souris et du clavier.

8.20 Référence du fichier view/dynamicElements/sourceview.hpp

```
#include <QGraphicsRectItem>
#include <QPen>
#include <QBrush>
```

Graphe des dépendances par inclusion de sourceview.hpp :



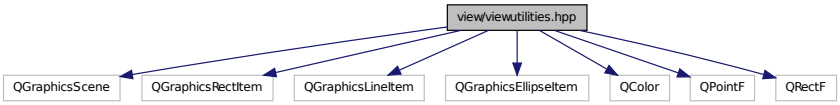
Classes

– class SourceView
Cette classe permet de représenter graphiquement une source, lui permettant de communiquer les actions utilisateurs.

8.21 Référence du fichier view/viewutilities.hpp

```
#include <QGraphicsScene>
#include <QGraphicsRectItem>
#include <QGraphicsLineItem>
#include <QGraphicsEllipseItem>
#include <QColor>
#include <QPointF>
#include <QRectF>
```

Graphe des dépendances par inclusion de viewutilities.hpp :



Espaces de nommage

– viewUtilities
Divers fonctions utilitaires nécessaires aux vues.

Fonctions

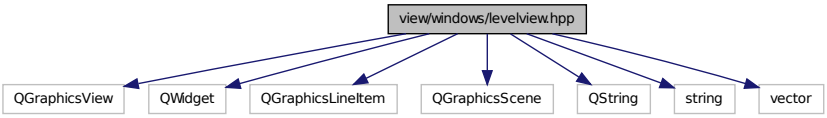
– QPointF viewUtilities : :toQPoint (const Point &)
Permet de transformer un point en QPointF.
– QRectF viewUtilities : :toQRectF (const Rectangle &)
Permet de transformer un rectangle en QRectF.
– QRectF viewUtilities : :toQRectF (const Ellipse &)
Permet de représenter une ellipse, à partir du rectangle qui lui est circonscrit, en un QRectF.
– QGraphicsLineItem ∗ viewUtilities : :getLine (const Point &, const Point &, const QColor &, const int)

Permet de générer une QGraphicsLine à partir des deux points délimitant un segment de droite.
– QGraphicsRectItem * viewUtilities : :getRect (const Rectangle &, const QColor &, const int)
Permet de générer un QGraphicsRectItem représentant le rectangle passé en paramètre.
– QGraphicsEllipseItem * viewUtilities : :getEllipse (const Ellipse &, const QColor &, const int)
Permet de générer un QGraphicsEllipseItem représentant l'ellipse passée en paramètre.
– QColor viewUtilities : :waveLengthToColor (const Ray &, const double=0.8)
Permet de créé une QColor au format RGB selon la longueur d'onde passée en paramètre.

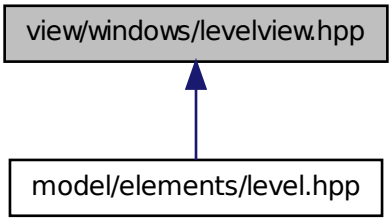
8.22 Référence du fichier view/windows/levelview.hpp

```
#include <QGraphicsView>
#include <QWidget>
#include <QGraphicsLineItem>
#include <QGraphicsScene>
#include <QString>
#include <string>
#include <vector>
```

Graphe des dépendances par inclusion de levelview.hpp :



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



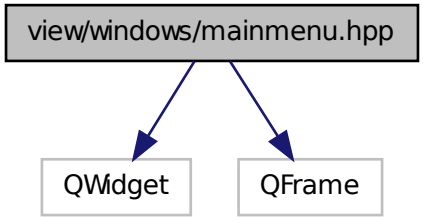
Classes

– class LevelView
Cette classe représente le niveau qui va être joué lors d'une partie.

8.23 Référence du fichier view/windows/mainmenu.hpp

```
#include <QWidget>
#include <QFrame>
```

Graphe des dépendances par inclusion de mainmenu.hpp :



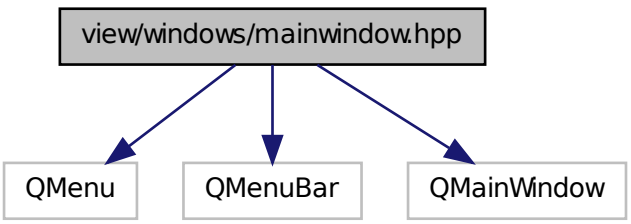
Classes

– class MainMenu
Cette classe représente le menu principal du jeu permettant de.

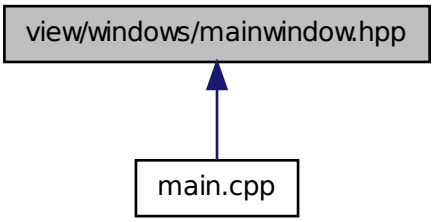
8.24 Référence du fichier view/windows/mainwindow.hpp

```
#include <QMenu>
#include <QMenuBar>
#include <QMainWindow>
```

Graphe des dépendances par inclusion de mainwindow.hpp :



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Classes

– class MainWindow
Cette classe est la fenêtre principale du jeu qui englobe toutes les autres vues.